2021

Digital volume correlation as a method for estimating load-induced deformations in the human spine

BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Dissertation

DIGITAL VOLUME CORRELATION AS A METHOD FOR ESTIMATING LOAD-INDUCED DEFORMATIONS IN THE HUMAN SPINE

by

JOHNFREDY LOAIZA

B.S., Brigham Young University, 2014
M.S., Boston University, 2019

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2021

Approved by

First Reader

Elise F. Morgan, Ph.D.
Professor of Mechanical Engineering
Professor of Materials Science and Engineering
Professor of Biomedical Engineering

Second Reader

Paul E. Barbone, Ph.D.
Professor of Mechanical Engineering
Professor of Materials Science and Engineering

Third Reader

Katherine Yanhang Zhang, Ph.D.
Professor of Mechanical Engineering
Professor of Biomedical Engineering
Professor of Materials Science and Engineering

Fourth Reader

Douglas P. Holmes, Ph.D.
Associate Professor of Mechanical Engineering
Associate Professor of Materials Science and Engineering

Fifth Reader

Michael B. Albro, Ph.D.
Assistant Professor of Mechanical Engineering
Assistant Professor of Materials Science and Engineering
Assistant Professor of Biomedical and Engineering

DEDICATION

To my darling wife Desiret and my wonderful children Benjamin, Elias, and Abraham.

ACKNOWLEDGMENTS

Thank you to Rohin Banerji, Dr. Paul Barbone, Dr. Glenn Barest, Joe Estano, Leslie Flores, Brian Gregor, Reece Huff, Dr. Amira Hussein, Krish Kapadia, Dr. Elise Morgan, Zack Webster, and Rachel Wilhelm for your substantial contributions to this work.

DIGITAL VOLUME CORRELATION AS A METHOD FOR ESTIMATING LOAD-INDUCED DEFORMATIONS IN THE HUMAN SPINE

JOHNFREDY LOAIZA

Boston University College of Engineering, 2021

Major Professor: Elise F. Morgan, Ph.D., Professor of Mechanical Engineering, Professor of Materials Science and Engineering, Professor of Biomedical Engineering

ABSTRACT

Digital volume correlation (DVC) is a computational tool used to measure a 3D displacement field between a pair of 3D images (from, for example, magnetic resonance imaging (MRI), computed tomography (CT), ultrasound, etc.). Studies in biomechanics have used DVC to quantify deformations in cells, tissues and organs, for the purpose of examining deformation and failure mechanisms, movement, and adaptation. The growing popularity of DVC has created increased demand for DVC algorithms that are computationally efficient, verified and validated. The goals of this project were to improve the efficiency of an existing DVC algorithm and to present a set of methods for robust verification and validation.

This dissertation first introduces DVC through a series of 1D examples that illustrates the use of optimization to find the displacement field that produces the best match between the pair of images. Different methods of regularization are explored. The concept of downsampling of the images is introduced as a way to promote faster convergence and a better image match.

In preparation for the move to 3D, the second part of the dissertation covers key

concepts of 3D image acquisition and data preparation for the specific case of μCT imaging of human vertebrae. This section allows the reader to appreciate the use of DVC to enable study of failure mechanisms in the spine.

The third section addresses the DVC method for 3D images. A custom process is introduced that uses rigid registration of the images to obtain an initial guess for the displacement field. The effect of the quality of the initial guess is then explored using test displacement fields.

In the final section, new methods of verification and validation of DVC are presented. An "image-warping" code is presented that interpolates a given displacement field to every voxel of an image, producing a synthetic image. This code is used to warp one image of a pair that was analyzed by DVC, and the mismatch between the synthetic image and the second image of the pair is used to verify the success of the minimization. The image-warping code is also used to create synthetic images from artificial, "test" displacement fields of increasing complexity and realism as a tool for validating the accuracy of the DVC algorithm. Finally, an L-curve method is applied in order to fine tune selection of the regularization parameter.

Though the improvements to DVC presented here were developed for the study of failure mechanisms of the spine, there is opportunity for broader application. The 1D examples can be mimicked to understand the foundations and limitations of similar DVC algorithms. Downsampling can also help these alternative algorithms to increase computational efficiency and improve image matching. Furthermore, the verification and validation methods presented here model an approach that others could use as they seek to

improve their own algorithms.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. DIGITAL VOLUME CORRELATION: A "BRIEF" OVERVIEW

1.1. Motivation

Digital volume correlation (DVC) is a computational tool developed to measure displacements fields from a series of images by tracking the intensity distribution throughout. Researchers have used DVC to measure 3D displacement fields in mechanical metamaterials, i.e. materials with structure, such as metal foams [1]–[3] and trabecular bone [3]–[8].

Bay et al. first introduced DVC in 1999 to measure rigid displacements (translation only, i.e. three degrees of freedom (DOF)) in trabecular bone scanned with X-ray tomography. Their algorithm has been advanced to solve for more degrees of freedom [1], [2], [8], [9], improve accuracy [6], [10], reduce computational cost [10], [11], and correct for noisy images [12], [13].

We have developed our own, in-house DVC algorithm to measure displacement fields in the human vertebral body with the ultimate goal of understanding the mechanical behavior of vertebral fractures. To promote reproducibility, this tutorial outlines the ins and outs of our algorithm.

1.2. DVC in 1D Case

1.2.1. Simple displacement derivation and introduction of a cost function

Let us imagine that we have a body in two configurations: before and after an arbitrary deformation. If we scan these two configurations of the body and obtain two perfect images (*e.g.*, no noise, infinitely small voxel size), then we can ideally map every voxel from one configuration to the other.

$$I_1(x) = I_2(x + u(x)) \tag{1.1}$$

$$0 = I_1(x) - I_2(x + u(x)) \tag{1.2}$$

Here, I_1 and I_2 represent the images before and after deformation, respectively, and $u(x)$ is the displacement field that will map the voxels from I_1 to I_2. Therefore, we wish to find an $u(x)$ that satisfies equation (1.1). However, because in reality the images will contain noise and have finite voxel size, we will find $u(x)$ using a cost function. In 1D, our cost function, C, is:

$$C(u(x)) = \frac{1}{2}\int_x \left[I_1(x) - I_2\big(x + u(x)\big)\right]^2 dx \tag{1.3}$$

We will minimize and then discretize C using the finite element mesh produced in section 3.3.

For now, let us consider a case where we know the function $u(x)$ that minimizes C. We will denote that $u(x)$ as $u^*(x)$ and consider a slight deviation from it:

$$C[u^*(x) + \beta w(x)] \tag{1.4}$$

where $w(x)$ is some arbitrary variation of u^* and β scales that variation. We substitute (1.4) into (1.3) and obtain

$$C[u^*(x) + \beta w(x)] = \frac{1}{2}\int_x \left[I_1(x) - I_2\big(x + u^*(x) + \beta w(x)\big)\right]^2 dx \tag{1.5}$$

We now define a function, $F^*(\beta)$:

$$F^*(\beta) = C[u^*(x) + \beta w(x)] \tag{1.6}$$

In equation (1.6), if we set $\beta = 0$, then we minimize C. Given that $F^*(\beta = 0)$ is a minimum, then its slope at $\beta = 0$ is zero:

$$\left.\frac{\delta F^*}{\delta \beta}\right|_{\beta=0} = 0 \tag{1.7}$$

Writing out the left-hand side of (1.7), this equation becomes:

$$\left.\frac{\delta F^*}{\delta \beta}\right|_{\beta=0} = -\int_x [I_1(x) - I_2(x + u^*(x))] \frac{\delta I_2}{\delta x}(x + u^*(x))\, w(x)\, dx = 0 \tag{1.8}$$

In principle, equation (1.8) can be solved for u^* and holds for any $w(x)$.

Let us now define a new function $F(u)$, where we replace $u^*(x)$ in equation (1.8) with $u(x)$:

$$F(u) = -\int_x [I_1(x) - I_2(x + u(x))] \frac{\delta I_2}{\delta x}(x + u(x))\, w(x)\, dx \tag{1.9}$$

Please note that our nomenclature "*" does not follow the analogy u is to u^* as *F(u)* is to $F^*(\beta)$. Also, from now on, we will simplify our notation of $u^*(x)$ to just u for convenience.

Our task can now be stated as "We wish to find u such that $F(u) = 0$."

For u, let us assume we have a "good" initial guess, u_0, i.e. a guess that is close to the u that minimizes equation (1.3), and that we are attempting to solve for an unknown, small displacement field, δu, which bridges u_0 to u:

$$u = u_0 + \delta u \tag{1.10}$$

We now substitute (1.10) into (1.9):

$$F(u_0 + \delta u) = -\int_x [I_1(x) - I_2(x + u_0 + \delta u)] \frac{\delta I_2}{\delta x}(x + u_0 + \delta u)\, w(x)\, dx \tag{1.11}$$

With the assumption that $\delta u \ll 1$ and using Taylor series expansions, we can write:

$$I_2(x + u_0 + \delta u) \approx I_2(x + u_0) + \frac{\delta I_2}{\delta x}(x + u_0)\delta u \tag{1.12}$$

$$\frac{\delta I_2}{\delta x}(x + u_0 + \delta u) \approx \frac{\delta I_2}{\delta x}(x + u_0) + \frac{\delta^2 I_2}{\delta x^2}(x + u_0)\delta u \tag{1.13}$$

We neglect the later Tayler expansion terms, because they include δu raised to a power greater than one.

By substituting (1.12) and (1.13) into (1.11), we get:

$$\begin{aligned} &F(u_0 + \delta u) \\ &= \int_x \left\{ \begin{array}{c} -[I_1 - I_2(x + u_0)]\frac{\delta I_2}{\delta x}(x + u_0) + \left(\frac{\delta I_2}{\delta x}(x + u_0)\right)^2 \delta u \\ + [I_1 - I_2(x + u_0)]\frac{\delta^2 I_2}{\delta x^2}(x + u_0)\delta u + O(\delta u^2) \end{array} \right\} w(x)dx \end{aligned} \tag{1.14}$$

We now make a Gauss-Newton approximation, which allows us to neglect the last two terms in the curly brackets of (1.14) by recognizing that $\frac{\delta^2 I_2}{\delta x^2}(x + u_0)$ and δu are both small quantities that, when multiplied together, result in a negligible quantity. Doing so makes (1.14) become:

$$\begin{aligned} F(u_0 + \delta u) = \int_x \Bigg\{ &-[I_1 - I_2(x + u_0)]\frac{\delta I_2}{\delta x}(x + u_0) \\ &+ \left(\frac{\delta I_2}{\delta x}(x + u_0)\right)^2 \delta u \Bigg\} w(x)dx \end{aligned} \tag{1.15}$$

Again, the approximations we have made only work if u_0 is close to u. Next, we set (1.15) equal to zero and rearrange:

$$\int_x \left\{ \left(\frac{\delta I_2}{\delta x}(x + u_0)\right)^2 \delta u \right\} w(x)dx = \int_x \left\{ [I_1 - I_2(x + u_0)]\frac{\delta I_2}{\delta x}(x + u_0) \right\} w(x)dx \tag{1.16}$$

By strategically choosing $w(x)$ to be nonzero over an infinitesimally small area, we can safely approximate (1.16) as:

$$\left\{\left(\frac{\delta I_2}{\delta x}(x+u_0)\right)^2 \delta u\right\}\int_x w(x)dx = \left\{[I_1 - I_2(x+u_0)]\frac{\delta I_2}{\delta x}(x+u_0)\right\}\int_x w(x)dx \tag{1.17}$$

This then allows us to eliminate $w(x)$:

$$\left(\frac{\delta I_2}{\delta x}(x+u_0)\right)^2 \delta u = [I_1 - I_2(x+u_0)]\frac{\delta I_2}{\delta x}(x+u_0) \tag{1.18}$$

Then, we solve for δu:

$$\delta u = \frac{[I_1 - I_2(x+u_0)]}{\frac{\delta I_2}{\delta x}(x+u_0)} \tag{1.19}$$

By recalling equation (1.10), we can now solve for $u(x)$.

Example 1: Calculating $u(x)$ for aligning 1D curves.

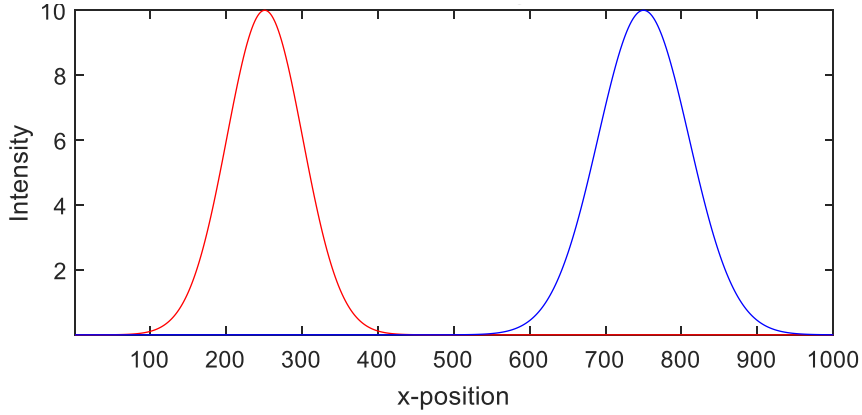


Figure 1.1: Initial 1D images, I_1 (red) and I_2 (blue),that we would like to match. I_2 is both offset by 500 "pixels" and horizontally stretched when compared to I_1. This introduces both rigid and non-rigid displacements.

As a working example, let us consider 1D images, I_1 and I_2, which are 1000-element arrays of intensity values ranging between zero and ten (see Figure 1.1). The main feature of these

images is a single peak of height (maximum intensity) = 10. In these examples, the pixel size is about 1/100th of the image feature size, and thus negligible. In I_1 this peak is centered at the 250th element of the array. In I_2 this peak is both offset from (the peak is now centered at 750th element) and wider than that of I_1. Our goal will be to calculate the displacement function u that will match I_2 to I_1. Here, we will use a constant initial guess, $u_0 = 480$, that will bring the peaks close to, but not completely aligned, each other:

Now, let us dive into the details of how $I_2(x + u_0)$ in equation 1.20 is obtained. For instance, the peak value of I_2, initially located at the 750th array element, is moved to the

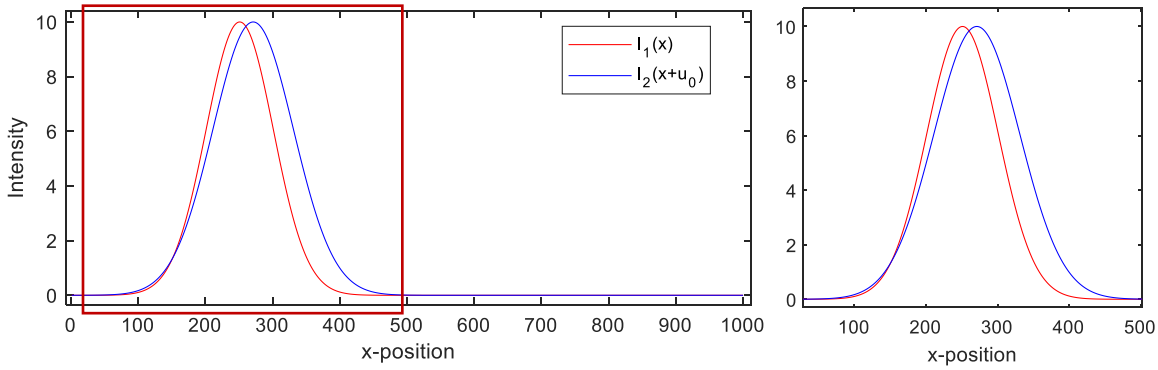


Figure 1.2: Plot of I_1 and I_2 after applying an initial guess for displacement. The plot on the left is cropped (right) to the non-zero portions of I_1 and I_2.

270th by populating a new, 1D array, I_2^{new}, defined as $I_2^{new}(x) = I_2(x + u_0)$, where x is the element number, *i.e.* index, and u_0 is the initial guess, which we have set to 480. Thus, $I_2^{new}(270) = I_2(270 + 480) = I_2(750)$. We follow this pattern for the entire I_2 array. However, whenever we index an array outside of its domain, we replace the produced "NaNs" with zeros; a step known as "zero padding". Having a constant u_0 effectively applies a strictly translational, no change in shape or size, displacement to the image that is represented by I_2.

Now, using equation (1.19), we calculate δu.

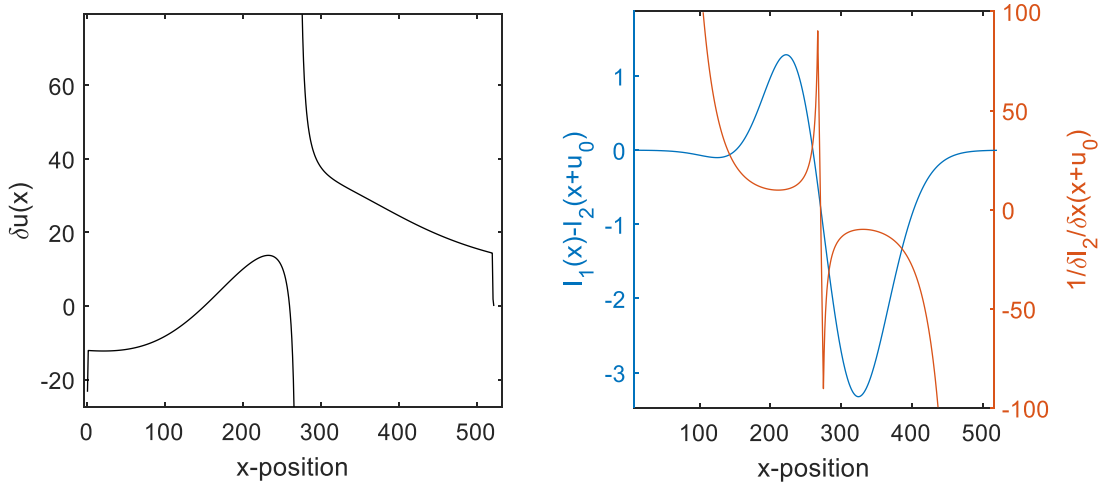


Figure 1.3: Plot of calculated $\delta u(x)$ (left) and the two curves, $I_1(x) - I_2(x + u_0)$ and $\left[\frac{dI_2}{dx}(x + u_0)\right]^{-1}$, that are multiplied to calculate $\delta u(x)$. Note that the non-physical spike seen in $\delta u(x)$ is caused by the spike in the inverse derivative of I_2.

Figure 1.3 (left) shows δu as a function of x. The key feature to note in this figure is the spike present at $x = 270$. This spike is caused by the zero slope at the peak of I_2, which in turn makes the reciprocal of this derivative equal to $1/0$ (infinity). In turn, the δu curve going to infinity means that we are indexing the "infinitieth" element of the I_2 array. Since this would produce a "NaN", we manually replace the "NaN" with a zero.

Regardless, we can see the quality of the match produced by using equations 1.20 and 1.21 by apply the δu to "warp", or resample, I_2 to create $I_{2'} = I_2(x + u_0 + \delta u(x))$:

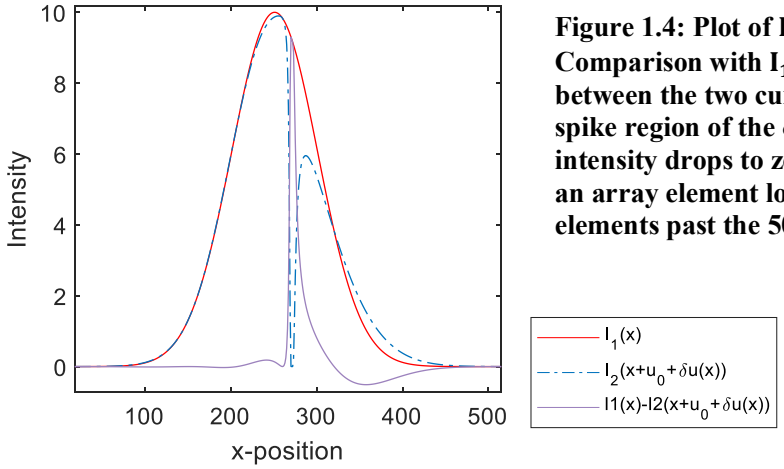


Figure 1.4: Plot of $\mathbf{I_2(x + u_0 + \delta u(x))}$. Comparison with $\mathbf{I_1}$ shows improved alignment between the two curves with the exception of the spike region of the δu (x=270). That region's intensity drops to zero because we sample $\mathbf{I_2}$ at an array element located at infinity (all array elements past the 500th are equal to 0).

Here, we see the trickle-down effect that the spike in the $\left(\frac{dI_2}{dx}\right)^{-1}$ has on the warped image. Besides that one region, the rest of the curves match fairly well. To quantify matching performance, we calculate a "mismatch" value, m:

$$m = \frac{\sum(I_1 - I_{2'})^2}{\sqrt{\sum I_1^2} * \sqrt{\sum I_{2'}^2}} \tag{1.20}$$

For the result shown in Figure 1.4, $m = 0.084$. That is, $I_{2'}$ is inaccurate by approximately 8%. We will compare this mismatch with other methods of determining δu, which we will detail next.

1.2.2. Displacement calculation with simple regularization

Moving on from example 1, we see that equation (1.19) presents problems whenever $\frac{\delta I_2(x+u_0)}{\delta x}$, the denominator, equals zero. To avoid this, we can use an arbitrary, positive definite parameter, α, in (1.18) and solve for δu:

$$\delta u = \frac{[I_1 - I_2(x + u_0)]\frac{\delta I_2(x + u_0)}{\delta x}}{\alpha + \left(\frac{\delta I_2(x + u_0)}{\delta x}\right)^2} \tag{1.21}$$

By maintaining the squared inverse derivate of I_2 in the denominator, we guarantee non-negative values in the denominator. The inclusion of positive α guarantees that the denominator will be positive definite; it will never be equal to or below zero.

Example 2: Matching single curve with α term.

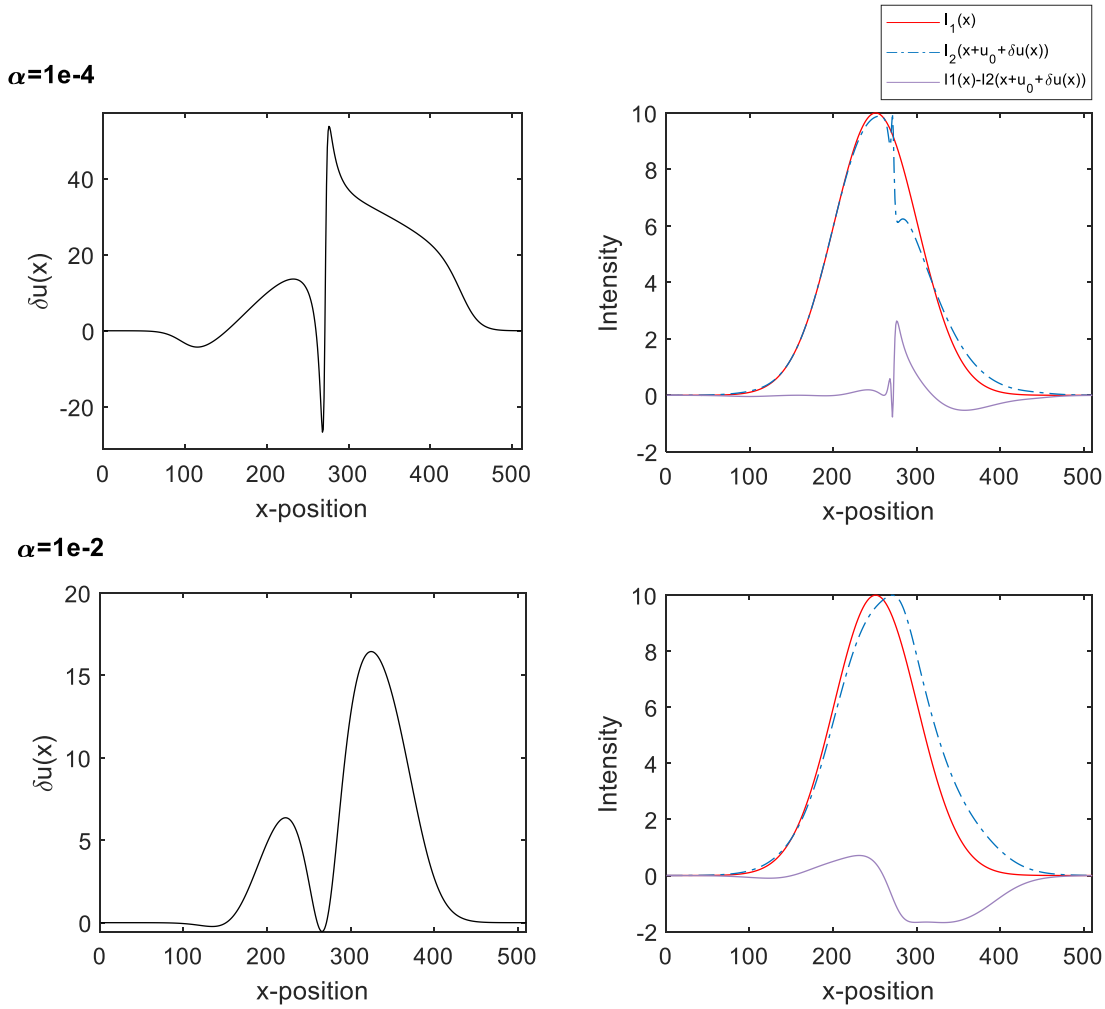


Figure 1.5: $\delta u(x)$ curve (left) and plot of $I_2(x + u_0 + \delta u(x))$ (right) when using term α in the δu calculation. When $\alpha = 1e-4$ (top), note that the spike behavior is still present in δu, but its peak values have been reduced. This results in a qualitatively better match than what we see without the α term. On the other hand, an α that is too high, i.e. $\alpha = 1e-2$ (bottom), offsets the δu curve enough to worsen the match.

We chose $\alpha = 1e-4$ such that it is just small enough to offset the squared inverse derivate of I_2 from zero (see top of Figure 1.5) without offsetting the rest of the displacement field by too much. The mismatch value is now reduced to $m = 0.012$.

When using a higher α ($\alpha = 1e-2$; see bottom of Figure 1.5), note that even though the $\delta u(x)$ curve becomes smoother (less of a spike present), the warped I_2 is lopsided and has poorer image matching ($m = 0.034$) than when the lower value of α was used. This smoothing achieved with α allows us to introduce the term "regularization", which is the process of correcting for regions in the domain where a unique finite solution is undefined. The "spike" is a region with no finite solution and through regularization, the addition of α, we seek to smooth over this region in the δu curve.

1.2.3. Regularization of displacement with spatial averaging

Besides the use of α, we can also smooth out δu through averaging. We will investigate three averaging schemes: global spatial averaging; local spatial averaging with *dis*continuous representation of δu; and local spatial averaging with continuous representation of δu. These averaging approaches differ in the size of averaged sub-regions, which we refer to here as span of array elements, and the interpolation function used to average across.

Example 3: Matching single curve with regularization through global spatial averaging.

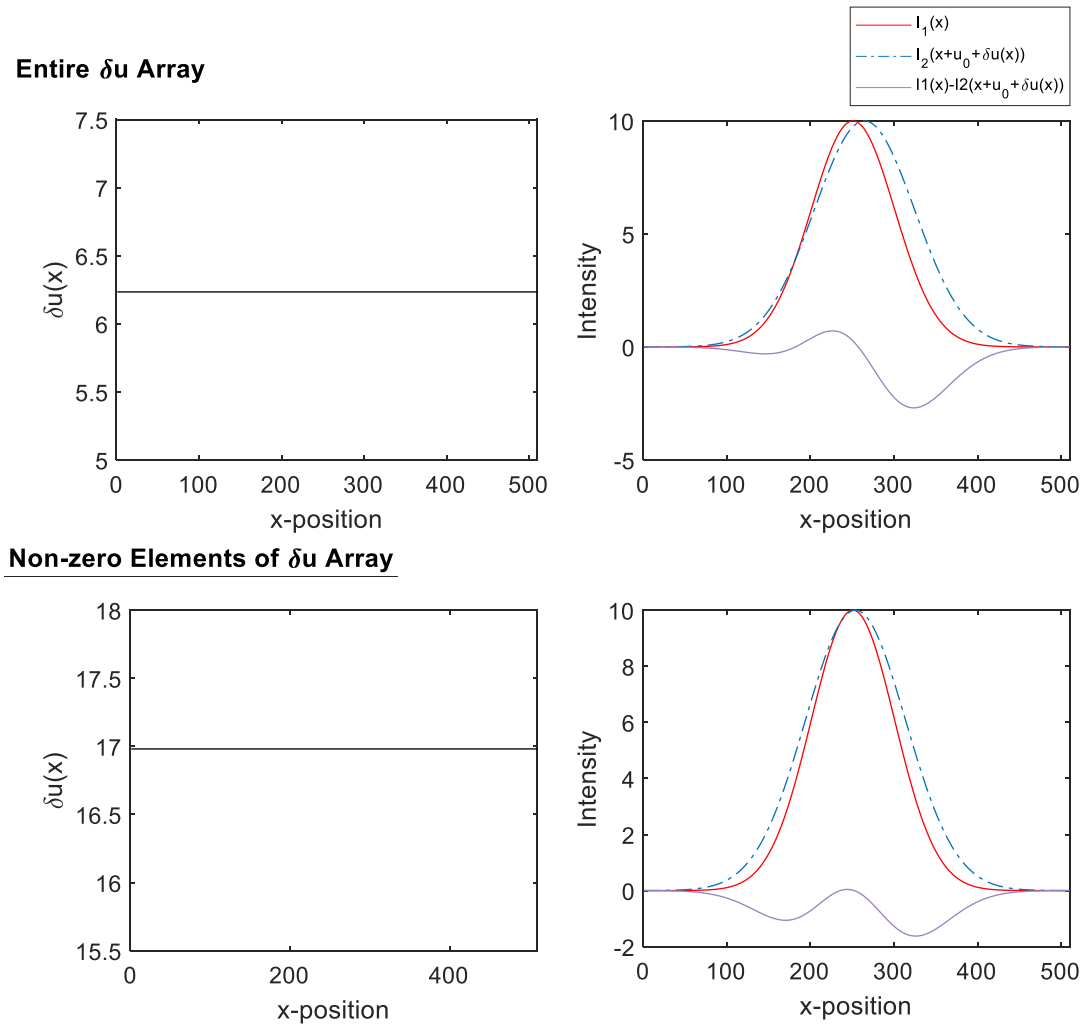


Figure 1.6: $\delta u(x)$ curve (left) and plot of $I_2(x + u_0 + \delta u(x))$ (right) when regularizing using the approach of global spatial averaging , where we average over the entire $\delta u(x)$ array (top) and the non-zero elements of the $\delta u(x)$ array (bottom). Note that I_2, with this approach, is rigidly moved, which brings the peaks of I_1 and I_2 closer, and poorly captures the non-rigid displacement.

With the first, global spatial averaging, we average across the entire array of δu and assume that δu is a constant term (see Figure 1.6). The resultant $I_{2'}$ rigidly approaches I_1 but cannot capture the non-rigid deformation that would exactly match I_1 . The mismatch

value in this case is $m = 0.055$. The residual misalignment of the peaks is due to the watering down effect of averaging a $\boldsymbol{\delta u}$ that has so many zeros. As a slight variation, we could limit our averaging span to the non-zero elements of δu (see bottom plots of Figure 1.6). This approach provides a much more accurate rigid displacement, but still cannot account for the non-rigid displacement ($m = 0.026$). Nevertheless, this improvement raises the question: is there a more appropriate averaging span that enables us to calculate the non-rigid displacements accurately?

Thus, we now explore local spatial averaging, which is averaging over smaller spans, or segments, of the $\boldsymbol{\delta u}$ array. To do so, we must change how we derive $\boldsymbol{\delta u}$, starting from equation (1.16). Now, instead of eliminating $w(x)$, we will define it as:

$$w(x) = \chi_B \tag{1.22}$$

where,

$$\chi_B = \begin{cases} 1 & x_B < x < x_{B+1} \\ 0 & otherwise \end{cases} \tag{1.23}$$

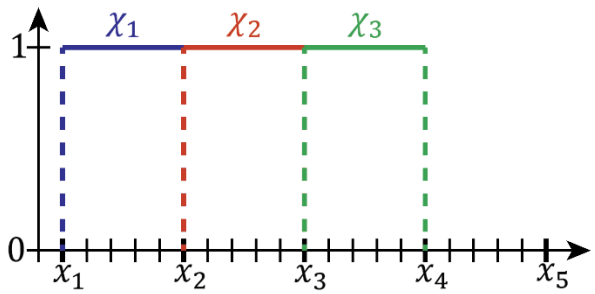


Figure 1.7: Depiction of three example χ_B functions. Each color represents a different span of χ.

Here, B is an indexing variable that steps through each individual span, or sub-region, of the $\boldsymbol{\delta u}$ array. For example, if we use an averaging span of fifty array elements, we index through every group of fifty elements. x_B and x_{B+1} represent the first and last elements of

each span. Figure 1.7 depicts a few indices of the χ_B function where each span is indicated in a different color.

We will also redefine $\boldsymbol{\delta u}$ in equation (1.16) as:

$$\boldsymbol{\delta u} = \sum_{A=1}^{n} \chi_A \delta u_A \tag{1.24}$$

A is also an indexing variable that steps through each span and n, the upper limit of the summation, is the total number of spans. For equation (1.23), subscript B is interchangeable with A. Here δu_A is a span, of $\boldsymbol{\delta u}$. Note that as we define smaller spans, n will become a larger value. To be more explicit,

$$n = \frac{\#\ of\ array\ elements\ in\ \boldsymbol{\delta u}}{\#\ of\ array\ elements\ in\ a\ span} \tag{1.25}$$

We now rewrite (1.16) by substituting (1.22)-(1.24):

$$\sum_A \int_x \chi_B \left(\frac{dI_2}{dx}\right)^2 \chi_A dx\, \delta u_A = \int_x \chi_B (I_1 - I_2) \frac{d_{I_2}}{dx} dx \tag{1.26}$$

Let us define the following:

$$K_{BA} = \int_x \chi_B \left(\frac{dI_2}{dx}\right)^2 \chi_A dx \tag{1.27}$$

$$F_B = \int_x \chi_B (I_1 - I_2) \frac{d_{I_2}}{dx} dx \tag{1.28}$$

Such that equation (1.26) can be rewritten as:

$$\sum_A K_{BA} \delta u_A = F_B \tag{1.29}$$

Equation (1.27) is simplified further if we recognize that:

$$K_{BA} = \begin{cases} K_{AA} & A = B \\ 0 & A \neq B \end{cases} \tag{1.30}$$

Thus, K_{AA} is a diagonal matrix, whose inverse is simply the inverse of each diagonal element. We can then solve (1.30) for δu_A:

$$\delta u_A = F_A / K_{AA} \tag{1.31}$$

Equation (1.31) is the calculation of $\boldsymbol{\delta u}$ with local spatial averaging, assuming discontinuous displacement, where we essentially calculate the average displacement for each span. With this approach, we can independently solve for each span of $\delta u(x)$, which allows computations to be performed in parallel.

Example 4: Matching single curve with regularization through local spatial averaging (assumed discontinuous displacement).

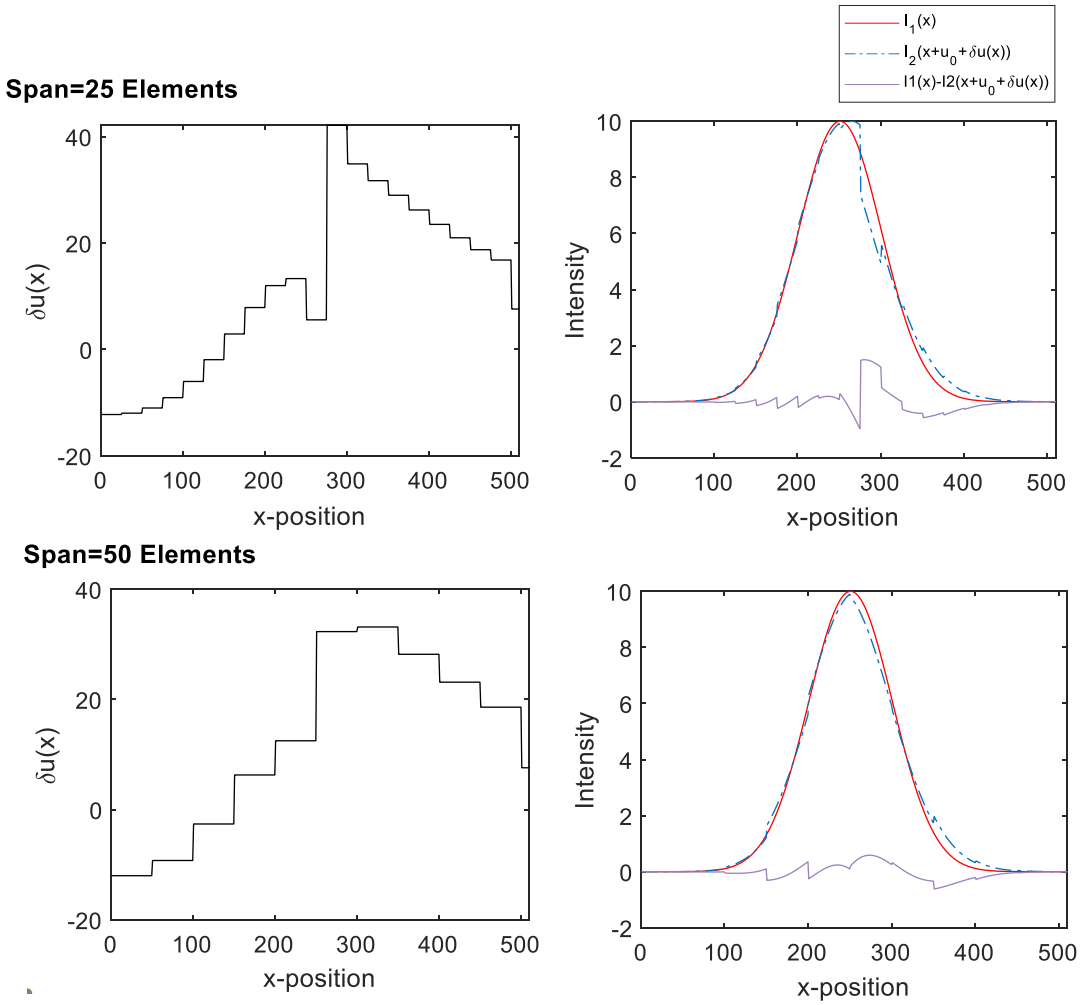


Figure 1.8: $\delta u(x)$ curve (left) and plot of $I_2(x + u_0 + \delta u(x))$ (right) when regularizing using the approach of local spatial averaging with a discontinuous displacement assumption , where we have a span length of 25 (top) and 50 (bottom) array elements.

The approach of local spatial averaging with assumed discontinuous displacement shows match improvement with an averaging span of 25 array elements (see Figure 1.8; $m = 0.009$) and more accurately captures the non-rigid displacements than global spatial

averaging. Increasing the averaging span to fifty array elements improves the match a little more ($m = 0.003$), though it is not safe too assume that simply increasing the span will always improve accuracy: an excessively large span would approach the results we saw with global spatial averaging (poor accuracy with non-rigid displacements). Instead, we conclude that for every displacement field u, there exists an ideal averaging span that accomplishes two things: 1) smooths over the spike region in the $\boldsymbol{\delta u}$ curve and 2) best matches the non-rigid displacements.

In the DVC field, local spatial averaging with discontinuous representation of the displacement field is known as "Local DVC", and the image match cost function typically used is cross-correlation [14], [15].

Let us now explore the performance of local spatial averaging with continuous representation of displacement. For this approach, we define a new $w(x)$ and $\boldsymbol{\delta u}$:

$$w(x) = N_B(x) \tag{1.32}$$

$$\boldsymbol{\delta u} = \sum_A N_A(x)\delta u_A(x) \tag{1.33}$$

Where,

$$N_B(x) = \begin{cases} \dfrac{x - x_{B-1}}{x_B - x_{B-1}} & x_{B-1} < x < x_B \\ \dfrac{x_{B+1} - x}{x_{B+1} - x_B} & x_B < x < x_{B+1} \\ 0 & otherwise \end{cases} \tag{1.34}$$

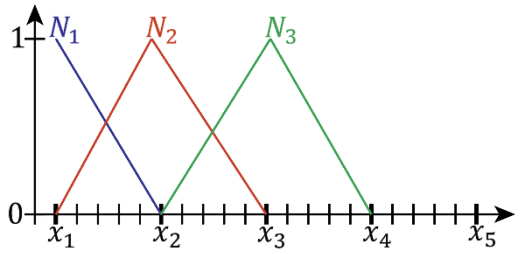


Figure 1.9: Depiction of three example N_B functions. Each color represents a different span of N_B.

We now substitute (1.32)-(1.34) into (1.16):

$$\sum_A \int_x N_B \left(\frac{dI_2}{dx}\right)^2 N_A dx\, \delta u_A = \int_x N_B (I_1 - I_2) \frac{dI_2}{dx} dx \tag{1.35}$$

We define new K_{BA} and F_B:

$$K_{BA} = \int_x N_B \left(\frac{dI_2}{dx}\right)^2 N_A dx \tag{1.36}$$

$$F_B = \int_x N_B (I_1 - I_2) \frac{d_{I_2}}{dx} dx \tag{1.37}$$

Such that,

$$\delta u_A = (K^{-1})_{BA} F_B \tag{1.38}$$

This approach has a higher computational cost than the approach that uses a discontinuous representation of δu, because of the need to calculate the inverse of K, which is a matrix whose size is inversely dependent on the averaging span; a smaller span will produce a larger matrix.

Example 5: Matching single curve with regularization through local spatial averaging (assumed continuous displacement).

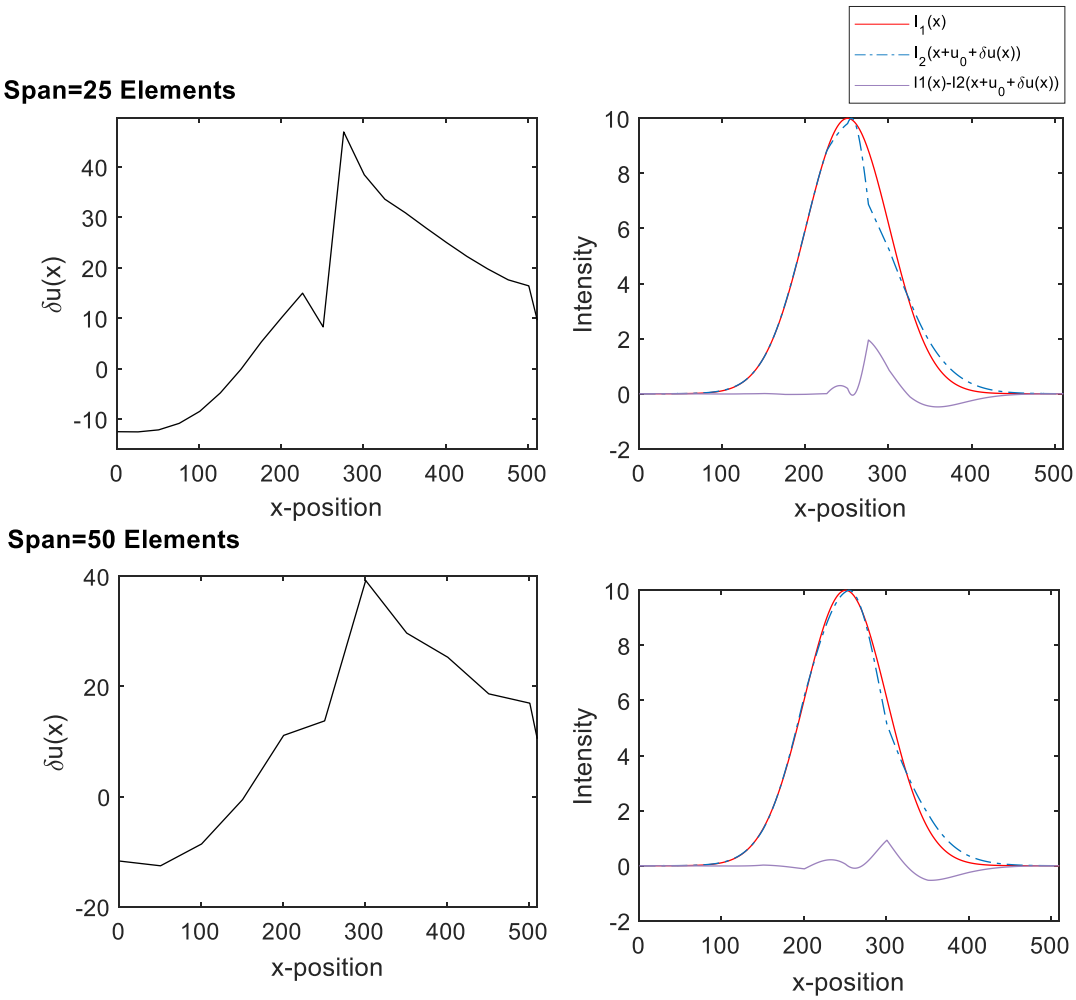


Figure 1.10: $\delta\mathbf{u}(\mathbf{x})$ curve (left) and plot of $\mathbf{I_2}\left(\mathbf{x}+\mathbf{u_0}+\delta\mathbf{u}(\mathbf{x})\right)$ (right) when regularizing using the local spatial averaging with a continuous representation of $\delta\mathbf{u}$, where we have a span length 25 (top) and 50 (bottom) array elements.

When comparing these two representations, discontinuous vs. continuous displacement, quantitative assessment shows little difference. For an averaging span of 25 elements, they produced mismatch values of $m = 0.009$ and $m = 0.011$, respectively (see Figure 1.10).

The 50-element averaging span produced mismatch values of $m = 0.003$ for both representations (for continuous displacement see the bottom row of plots in Figure 1.10).

The difference between the two is instead visible in the qualitative review of data. As expected, the continuous displacement representation provides a smoother $I_{2'}$ (compare Figure 1.9 to Figure 1.10). Still, depending on the application, either approach could provide a sufficient solution. Our goal however, is to achieve better accuracy in both quantitative and qualitative measures.

1.2.4. Iterative method for calculating displacement

To arrive at a more accurate solution, we explore an iterative method. In this method we:

1) Apply an initial guess, u_0, and solve for δu, as we have done above. This δu we denote as δu_1.
2) Create a new initial guess, $u_{0_2} = \delta u_1 + u_0$ and solve for δu as we have done above. This δu we denote as δu_2.
3) For iteration $p = 3$ and onward, use $u_{0_p} = \sum_{i=1}^{p-1} \delta u_i + u_0$ as the initial guess and solve for δu_p.
4) Stop when the δu_p update becomes small.

To determine the stopping point in step 4, we calculate the root mean square of the each displacement update, δu_p, as:

$$RMS_{\delta u_p} = \sqrt{\frac{\boldsymbol{\delta u_p} \cdot \boldsymbol{\delta u_p}}{n}} \tag{1.39}$$

Where scalar n is the length (number of elements) of the δu_p array. We cut off our iterations when $RMS_{\delta u_p}$ either reaches a low, user-defined value or has a low gradient. Let us compare performance of both local spatial averaging approaches when iteratively approaching a solution.

Example 6: Matching single curve iteratively with regularization through local spatial averaging (both displacement representations).

Figure 1.11 shows a clear winner in achieving high displacement accuracy. The first quartet of plots demonstrate that after ten iterations, a discontinuous representation, will settle at a low-$RMS_{\delta u}$ and still not achieve the accuracy seen in the following quartet, which achieves near-perfect accuracy in just five iterations (when it reaches a similarly low $RMS_{\delta u}$). The higher accuracy is not only evident in the image match, but also the mismatch plots. The fact that a continuous displacement is present in the original graphs is why a continuous representation provides the most accurate match.

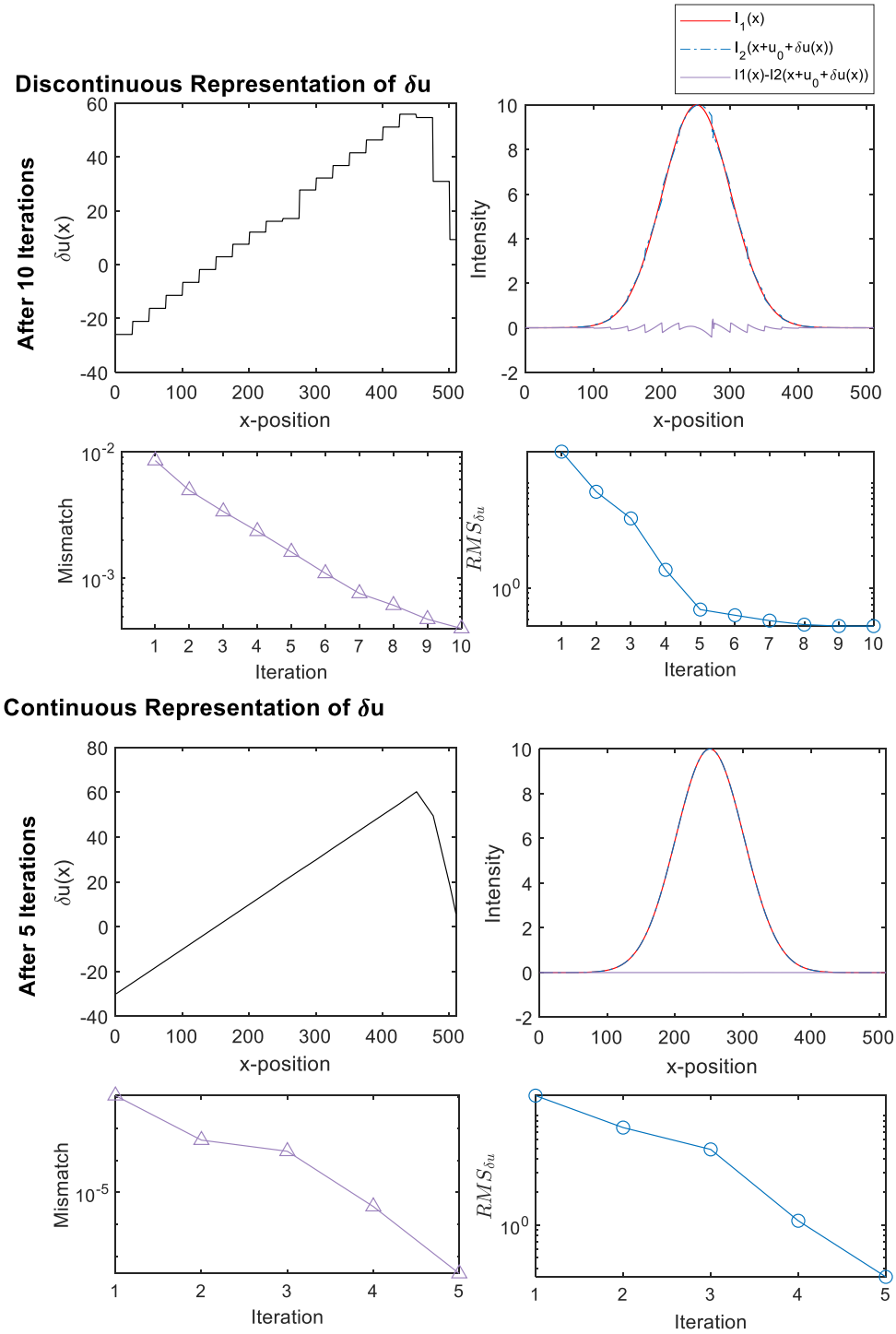


Figure 1.11: Iterative performance of the discontinuous (top) and continuous (bottom) representation of displacement. For each quartet of plots, the $\delta u(x)$ curve is found on the top left and the plot of $I_2(x + u_0 + \delta u(x))$ on the top right. We also see measures of mismatch (bottom left) and $RMS_{\delta u}$ (bottom right) at each iteration. Where the discontinuous representation of δu took ten iterations to arrive at a low $RMS_{\delta u}$, the continuous representation only took five.

1.2.5. Determination of an appropriate initial guess

Now that we have reviewed the iterative method, we explore the question: "How close does the initial guess need to be for convergence to occur?" In this exploration, we will limit ourselves to a constant initial guess, i.e. a rigid displacement, and use the continuous displacement representation, as this is what we use in our full DVC algorithm.

Example 7: Exploration of an appropriate initial guess.

First, let us see what happens when our initial guess places our peaks twice as far from each other than before (see Figure 1.2 for a "better" initial guess) and use an averaging span of 25 array elements. In this instance (see Figure 1.12), the spike behavior reappears at the first iteration and it takes the local spatial averaging with continuous representation algorithm thirteen iterations to match the two peaks. The reappearance of the spike behavior is caused by the fact that our averaging span does not bridge across this troublesome region. What we can do to improve performance is use a larger span of fifty array elements.

With a larger averaging span (see Figure 1.13), we successfully average across the spike in the δu curve. Also, with the larger averaging span, image match occurs in as little as five iterations. Both Figure 1.12 and Figure 1.13 demonstrate the balancing act that is required between peak distance and averaging span length to assess a good initial guess. We find the cause for this behavior to be in equation (1.19), which simply states that δu is the ratio of image difference over the derivative of I_2. A further peak distance will increase image difference and a smaller averaging span length will fail to bridge over the spike

region. If we couple these two effects (as seen in Figure 1.12), we see that an increase in image difference multiplied by the increased spike region greatly increases the spike seen in the image match and the number of iterations required to arrive at the solution. Therefore, if we increase averaging span, we reduce the spike behavior and enable faster convergence (as seen in Figure 1.13). Theoretically, at a minimum, the two peak tails would only need to overlap so that we can eventually arrive at the correct solution. However, as we have learned, such distant peaks would require an inordinate amount of iterations.

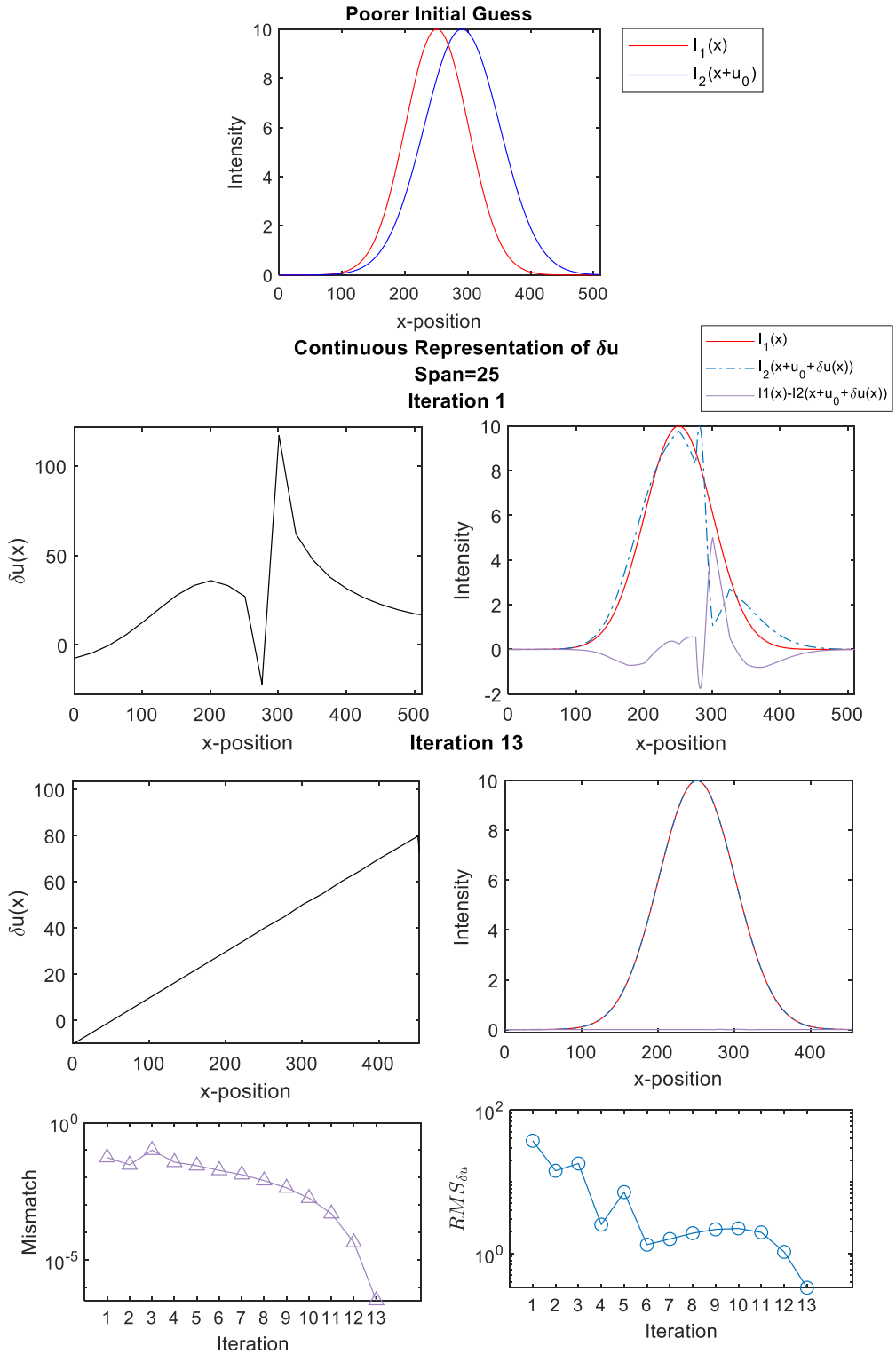


Figure 1.12: First (middle) and thirteenth (bottom) iteration performance when starting with a poorer initial guess (top) of local spatial averaging with continuous representation of δu when averaging across twenty-five array elements.

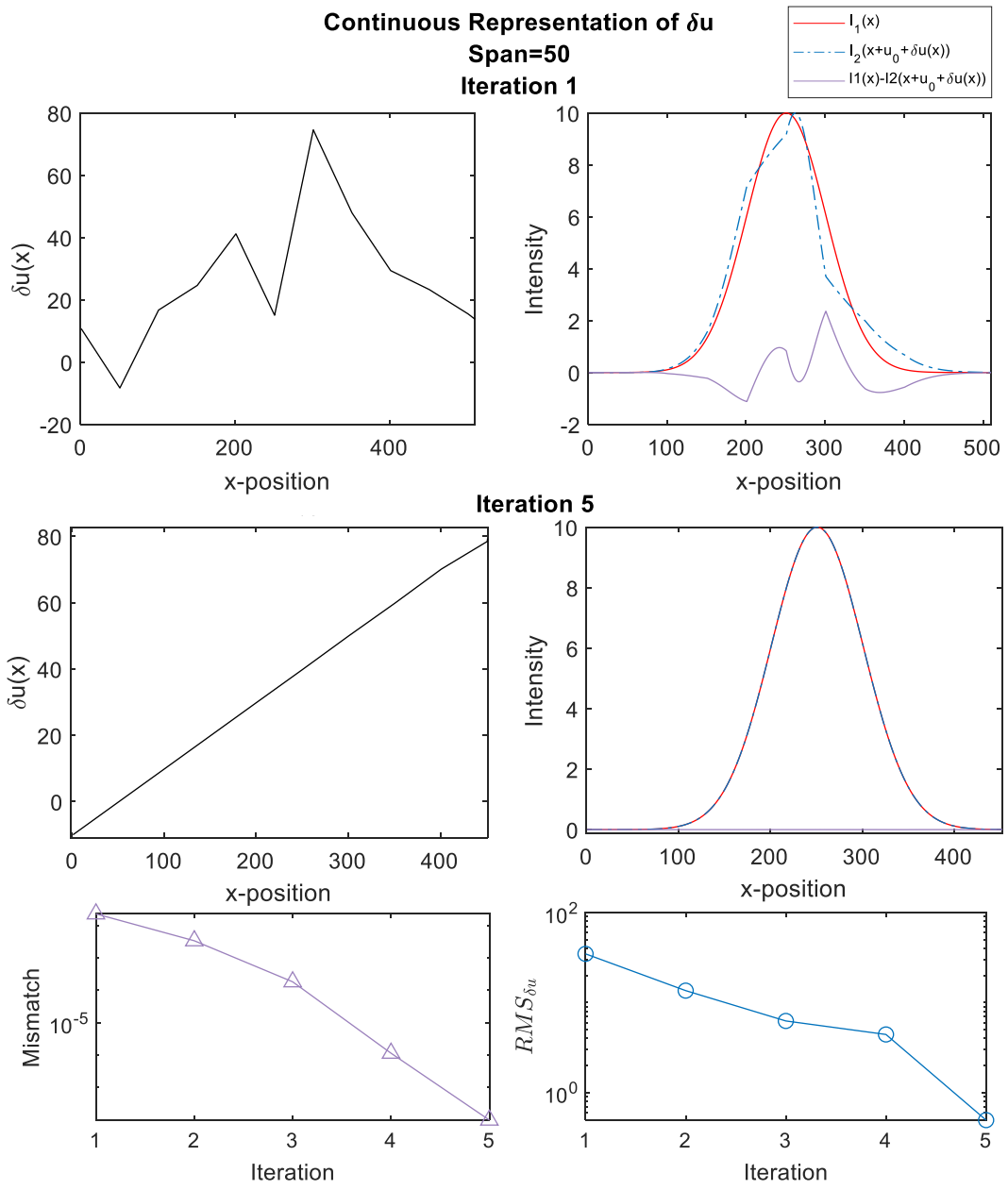


Figure 1.13: First (top) and fifth (middle) iteration performance when starting with a poorer initial guess of local spatial averaging with continuous representation when averaging across fifty array elements.

1.2.6. Regularization to correct for noise

Since noise is always present in any imaging modality (MRI, CT, or Ultrasound), even with a good initial guess, using local spatial averaging with a continuous representation as presented is insufficient for regularizing all of the peaks that appear in a noisy curve (see Figure 1.14).

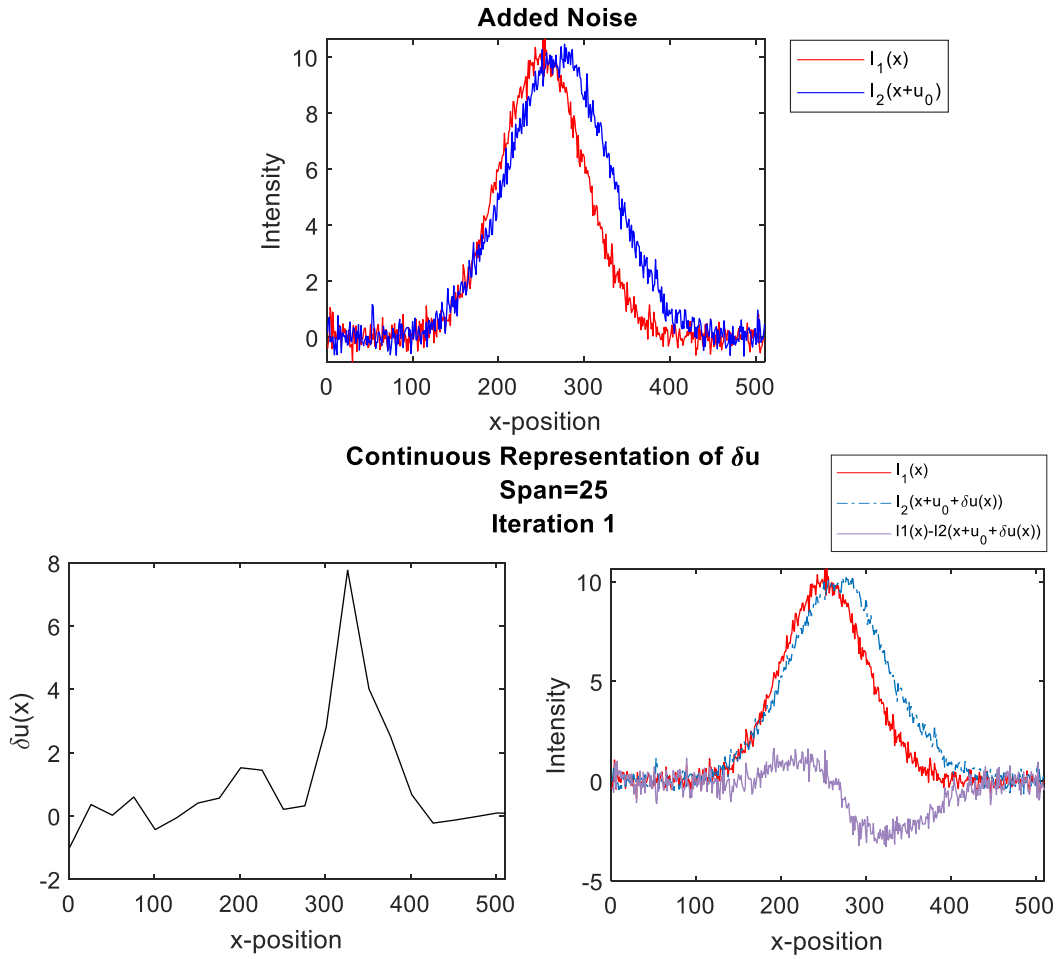


Figure 1.14: Peaks of I_1 and I_2 after applying noise (top). As a result, the δu curve (bottom left) using local spatial averaging with continuous representation of δu contains noise. The plot of $I_2(x + u_0 + \delta u(x))$ (bottom right) shows small improvement in image matching after one iteration.

To address this limitation, we enhance the cost function presented in equation (1.3) and revisit the addition of the regularization term α:

$$C\big(u(x)\big) = \frac{1}{2}\int_x \big[I_1(x) - I_2\big(x + u(x)\big)\big]^2 dx + \frac{\alpha}{2}\int_x \left(\frac{du}{dx}\right)^2 dx \tag{1.40}$$

With the addition of the second term in equation (1.40), we limit large displacement gradients. In other words, we prevent sudden jolts in the δu curve. With a continuous representation of displacement, we can follow a similar procedure used to derive equations (1.36) and (1.37) to calculate K_{BA} and F_B that include the second term in equation (1.40):

$$K_{BA} = \int_x N_B \left(\frac{dI_2}{dx}\right)^2 N_A dx + \alpha \int_x \frac{dN_A}{dx}\frac{dN_B}{dx} dx \tag{1.41}$$

$$F_B = \int_x N_B (I_1 - I_2)\frac{d_{I_2}}{dx} dx - \alpha \int_x \frac{du_0}{dx}\frac{dN_B}{dx} dx \tag{1.42}$$

We solve for δu by subbing equations (1.41) and (1.42) into equation (1.38).

Example 8: Matching a single curve with local spatial averaging (continuous representation) and a regularization term, α.

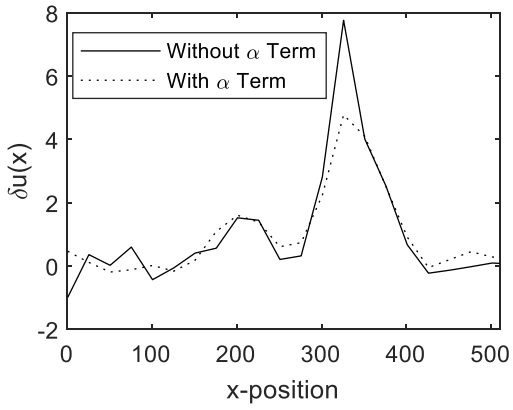


Figure 1.15: δu smoothness comparison when calculated with (α = 10) and without the α term in the cost function.

For this example, we use an averaging span of 25 array elements and set α equal to ten. After one iteration, the new δu curve is smoother (see Figure 1.15).

Where we truly reap the benefits of this compound regularization approach (local spatial averaging and α) is when we wish to

decrease our averaging span to favor spatial resolution without sacrificing accuracy in our final δu function. To demonstrate this, we will reduce our averaging span to only ten array elements and continue to use an α of ten.

First, let us review the performance of the continuous representation with a ten-element averaging span, without the α term. Noise is much more visible in δu_1, δu from the first iteration, (see Figure 1.16) and continues to persist after 25 iterations. Still, the images appear to match well and converge to a low mismatch value ($m(25) = 0.016$). This teaches us that even though our images qualitatively and quantitatively match well, when the averaging span is small and there is noise present, we misrepresent the δu curve; the true displacement is not as "jolty". This causes the jittery development of the $RMS_{\delta u}$ plot.

With the inclusion of α, we can smooth out the δu curve without sacrificing too much in accuracy. The smoothing is immediately evident in the first iteration (see Figure 1.17) and becomes more so after 25 iterations. This causes the smoother convergence of $RMS_{\delta u}$. The δu curve of the twenty-fifth iteration has a slightly poorer matching behavior and lower mismatch value ($m(25) = 0.020$) than the previous, noisier approach. However, we see improvement in a δu that is more representative of the true, smooth displacement.

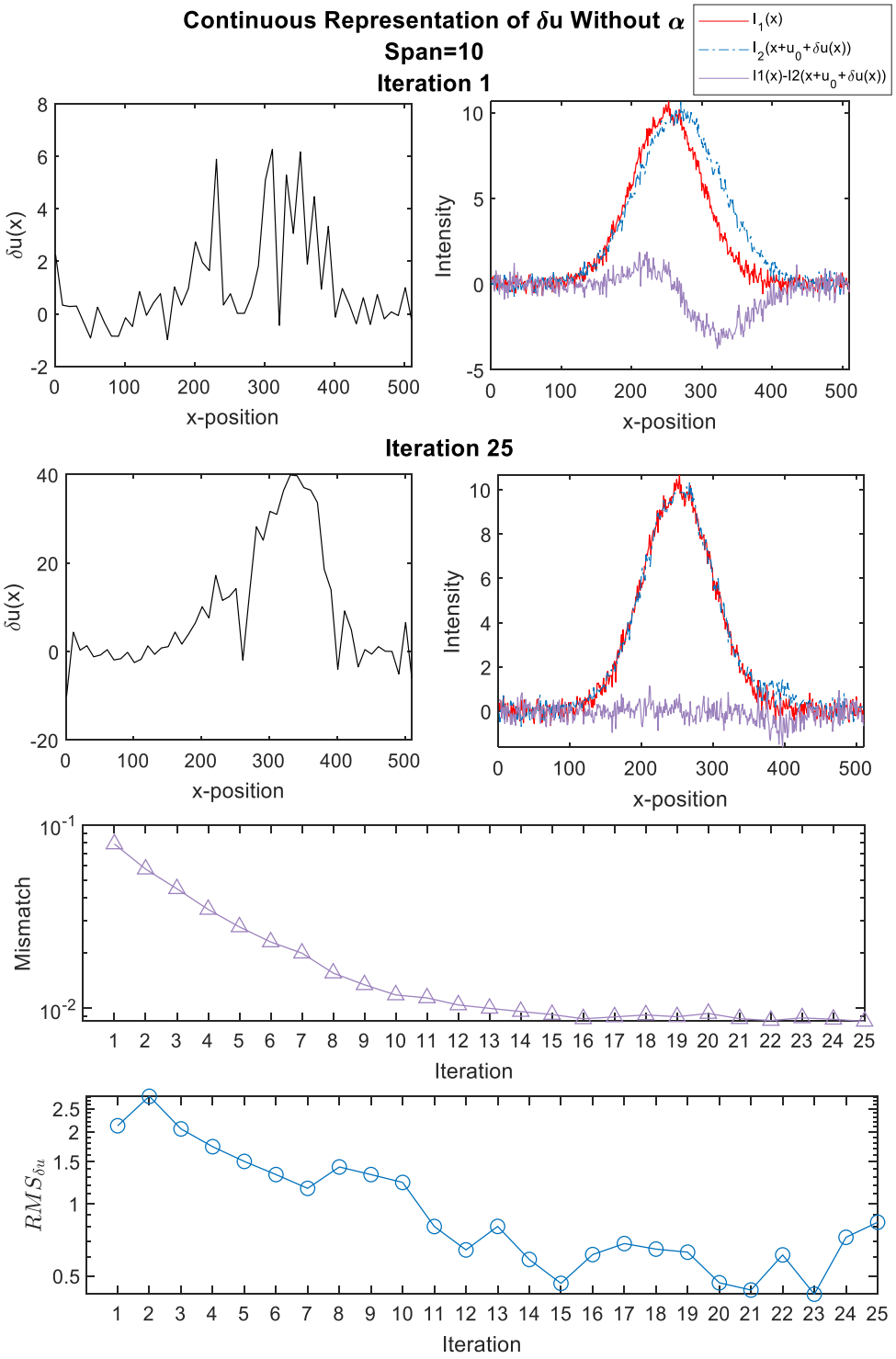


Figure 1.16: Iterative performance of spatial averaging with continuous representation of δu (no α) when there is noise present in the curves. A span of ten array elements was used. The first (top) and twenty-fifth (second row) iterations are plotted above. In addition, the mismatch (second to last plot) and $RMS_{\delta u}$ (bottom) plots depict convergence behavior.

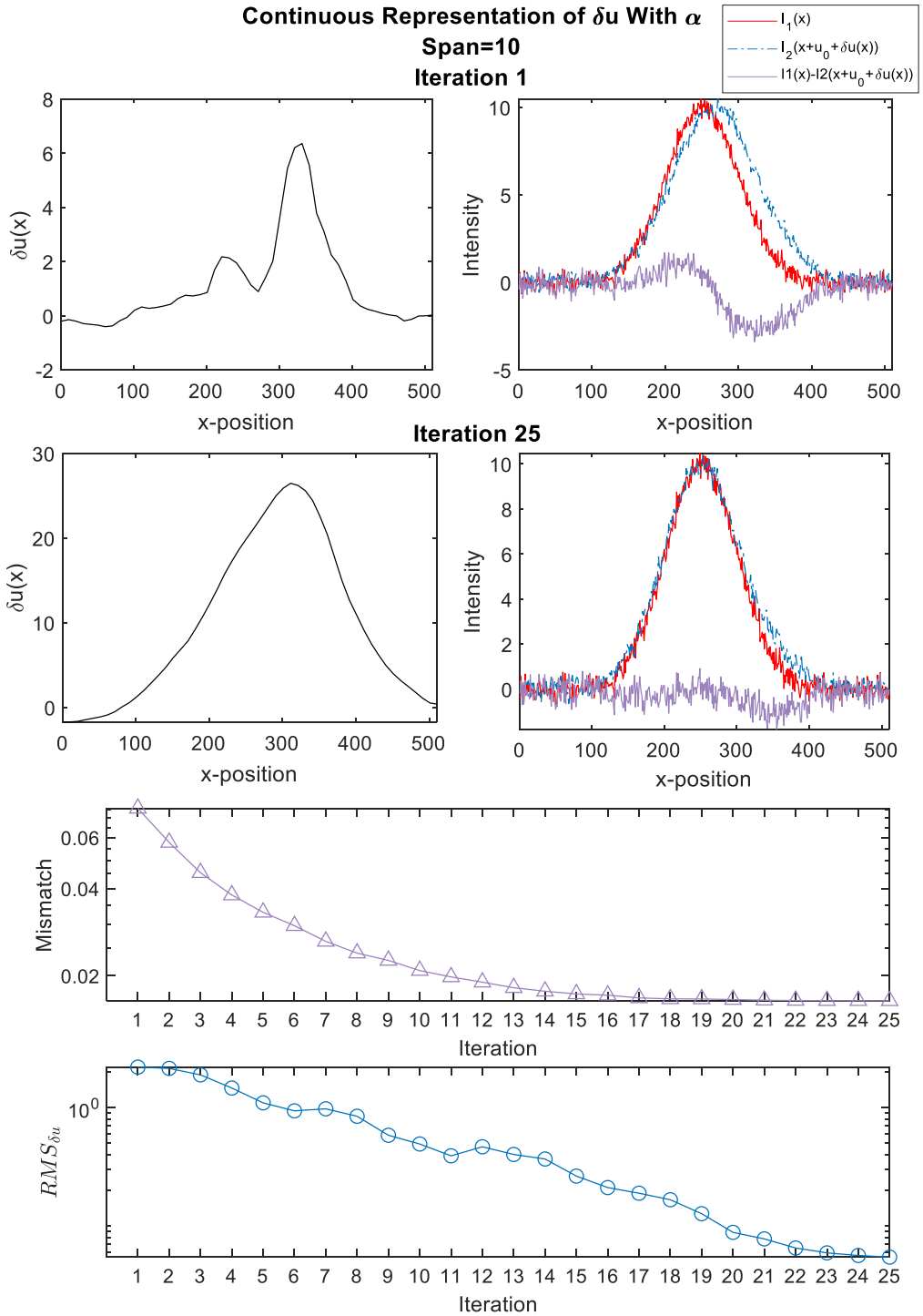


Figure 1.17: Iterative performance of spatial averaging with continuous representation of δu (with α = 10) when there is noise present in the curves. A span of ten array elements was used. The first (top) and twenty-fifth (second row) iterations are plotted above. In addition, the mismatch (second to last plot) and $RMS_{\delta u}$ (bottom) plots depict convergence behavior.

1.2.7. Downsampling to avoid convergence on a local minimum due to a poor initial guess

To conclude our examples, we will look at situations where DVC can converge on a local minimum. To depict this, we will change our curves that consist of one peak to six each and only apply a rigid displacement of thirty array elements between the two curves (see Figure 1.18). We first used a rigid displacement approach (see Figure 1.19) by applying a large averaging span (span = 500) with a discontinuous representation of δu. Doing so causes the algorithm to approach and settle on an incorrect solution: after eleven iterations, we have aligned the wrong peaks to each other. This is convergence to a local minimum. Even using a continuous displacement with regularization in the cost function (see Figure 1.20), which allows for non-rigid displacement, the algorithm fails to approach a correct solution after seventeen iterations. This behavior occurs because the wrong peaks are close to each other.

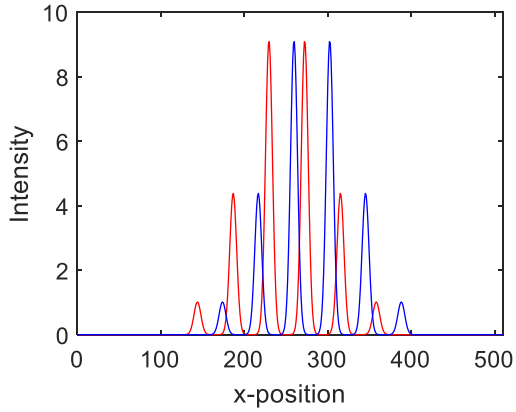


Figure 1.18: I_1 and I_2, both with six peaks instead of just one, begin rigidly offset from one another.

To avoid this behavior, we smooth out the curves using a downsampling algorithm, which essentially conglomerates the peaks into a single, though oddly-shaped, peak (see Figure 1.21). Doing so eliminates the possibility of falsely placing the wrong peaks beside each other. With downsampling, we calculate a displacement field that settles at approximately

thirty array elements: the initial, applied rigid displacement.

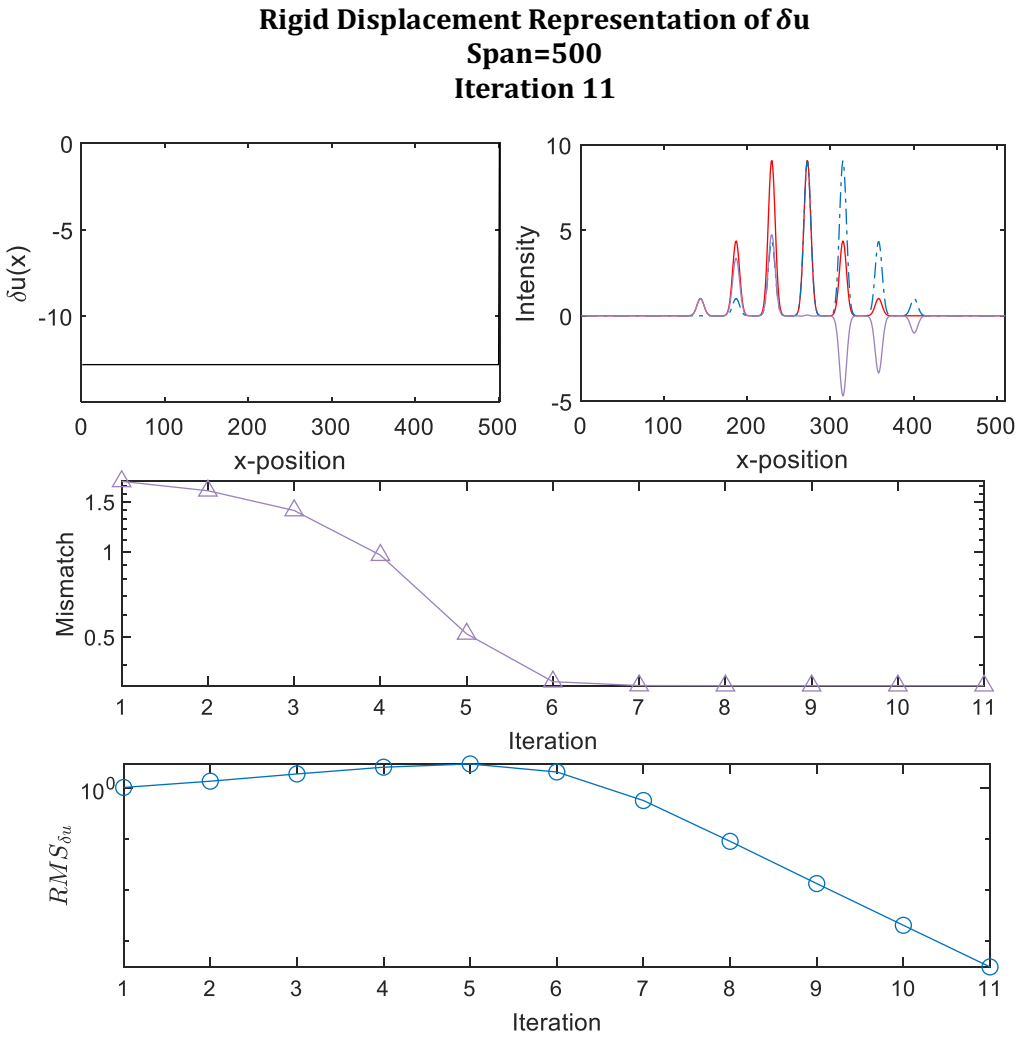


Figure 1.19: Performance of the rigid displacement representation of δu with multiple peaks and a poor initial guess. After eleven iterations, the low gradient of the $RMS_{\delta u}$ curve (bottom) demonstrates convergence, but the intensity plot of $I_2(x + u_0 + \delta u(x))$ (second row) and mismatch plots (second to last plot) depict poor alignment.

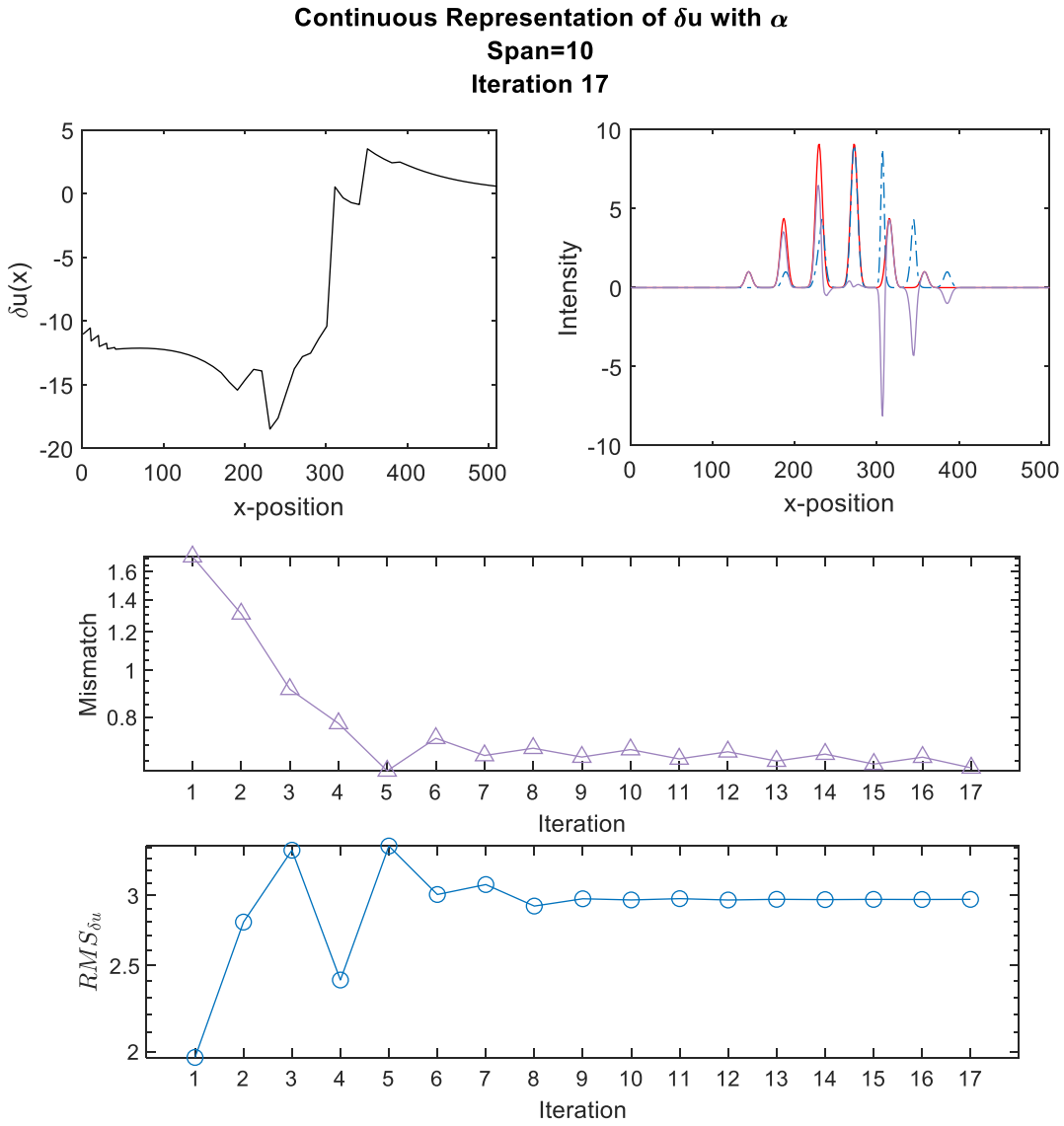


Figure 1.20: Performance of the continuous representation of δu (with α) with multiple peaks and a poor initial guess. After seventeen iterations, the low gradient of the $RMS_{\delta u}$ curve (bottom) demonstrates convergence, but both the of $I_2(x + u_0 + \delta u(x))$ (second row) and mismatch plots (second to last plot) depict poor alignment.

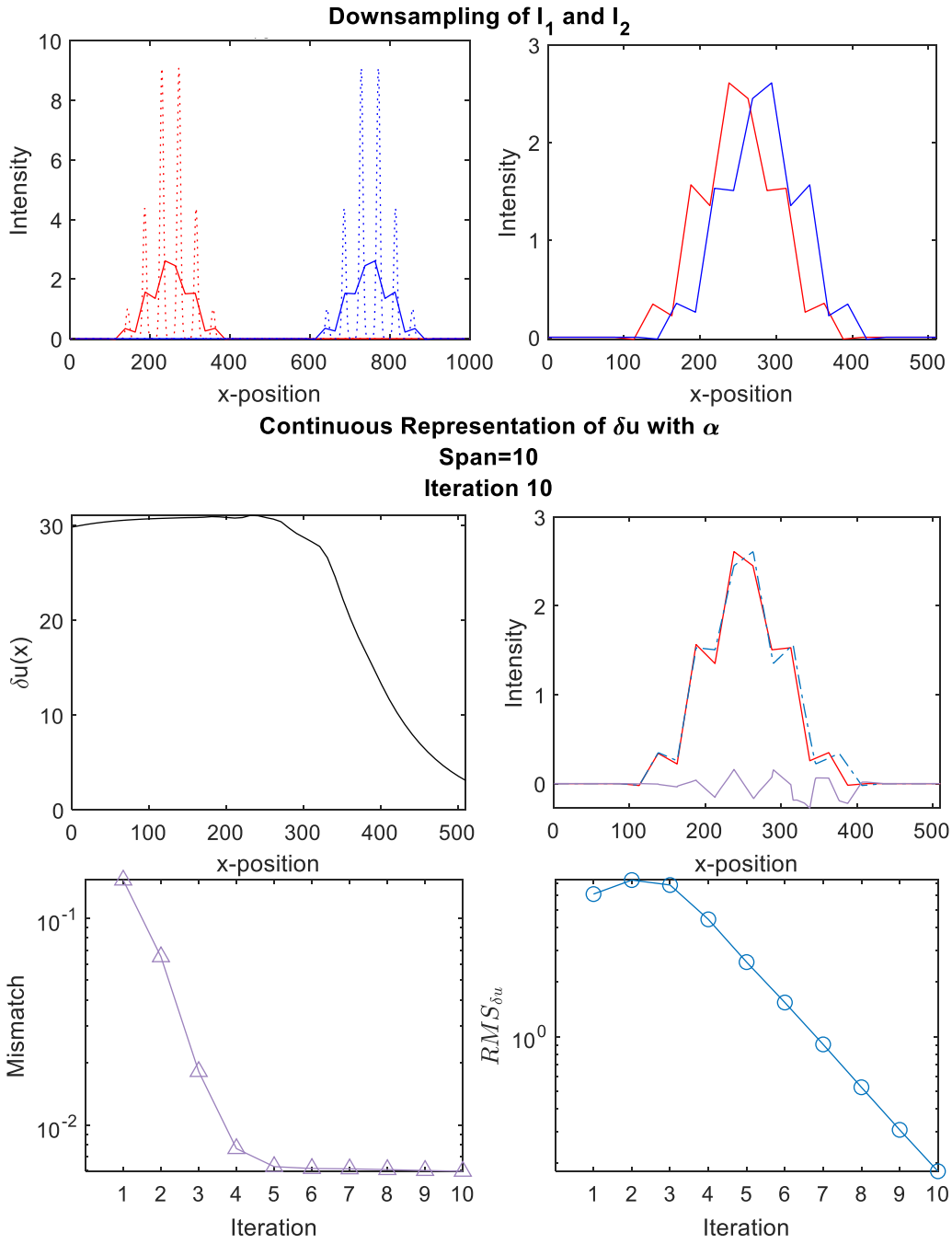


Figure 1.21: Performance of the continuous representation of δu (with α) with multiple peaks and a poor initial guess is improved when downsampling. I_1 and I_2 (top left) become averaged down with a downsampling algorithm prior to applying the constant initial guess (applied in top right). This provides a smooth δu curve (middle left) and a $I_2(x + u_0 + \delta u(x))$ that visibly aligns well with I_1 (middle right). After ten iterations, there is clear convergence in the $RMS_{\delta u}$ curve (bottom right). The mismatch plot (bottom left) depicts great alignment.

1.2.8. Strain calculation in 1D

We calculate strain for each span, ϵ_A, using a strain-displacement vector, B_A, and displacement vector as so:

$$\epsilon_A = B_A \begin{bmatrix} u_A \\ u_{A+1} \end{bmatrix} \tag{1.43}$$

where B_A, not be confused with the subscript B from above, in this 1D example is:

$$B_A = \begin{bmatrix} \frac{dN_A}{dx} & \frac{dN_{A+1}}{dx} \end{bmatrix} \tag{1.44}$$

With equation (1.44), we must take care to use the portion of the shape functions that lie within the span of interest. For example, let us look at what B_1 looks like:

$$B_1 = \begin{bmatrix} \frac{dN_1}{dx} & \frac{dN_2}{dx} \end{bmatrix} = \begin{bmatrix} \frac{-1}{x_2 - x_1} & \frac{1}{x_2 - x_1} \end{bmatrix} \tag{1.45}$$

Keeping this example in mind, we rewrite equation (1.43):

$$\epsilon_A = \begin{bmatrix} \frac{1}{x_{A+1} - x_A} \end{bmatrix} [-1 \quad 1] \begin{bmatrix} u_A \\ u_{A+1} \end{bmatrix} = \frac{u_{A+1} - u_A}{x_{A+1} - x_A} \tag{1.46}$$

1.2.9. Summary

In summary, we have presented an algorithm that will iteratively calculate the displacement field, u, that aligns a deformed, 1D image to its original undeformed position. This algorithm includes regularization through spatial averaging and an extra regularization term in the cost function. Doing so allows us to arrive at an accurate measure of displacement regardless of noise present. In addition, our use of a continuous representation of displacement enables us to present the most accurate solution. We address the algorithm's primary pitfall of settling to a local minimum with downsampling. Moving forward, we present how we apply these concepts to calculate the displacement field of mechanically deformed vertebrae captured with 3D, μCT images.

1.3. Full DVC Flow

In our DVC algorithm, we leverage the concepts presented in the previous section: regularization in the cost function and through spatial averaging with a continuous representation of displacement; a custom algorithm for finding an appropriate initial guess; and downsampling to enable convergence on the global minimum. Below, we include both overview (see Figure 1.22) and detailed (see Figure 1.23), where we include algorithm names, flowcharts for reference. In the following chapters, we provide detail into the non-trivial steps of the algorithm.

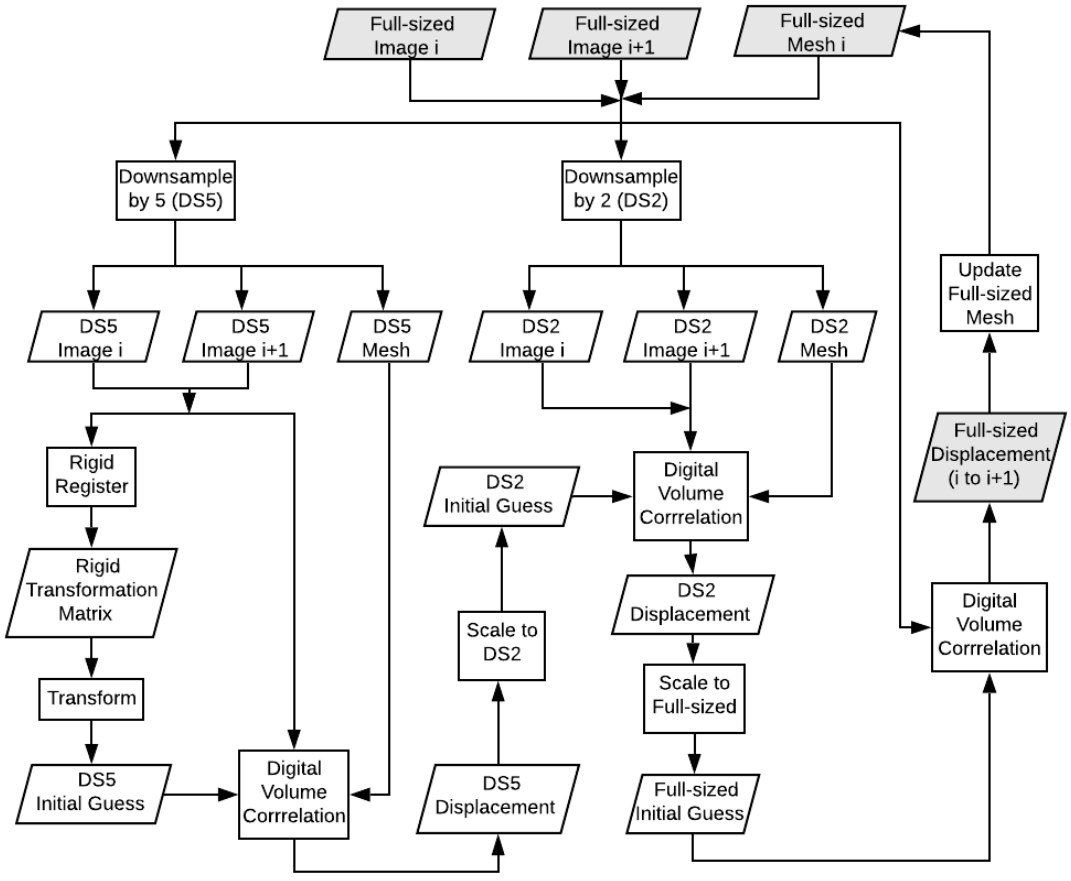
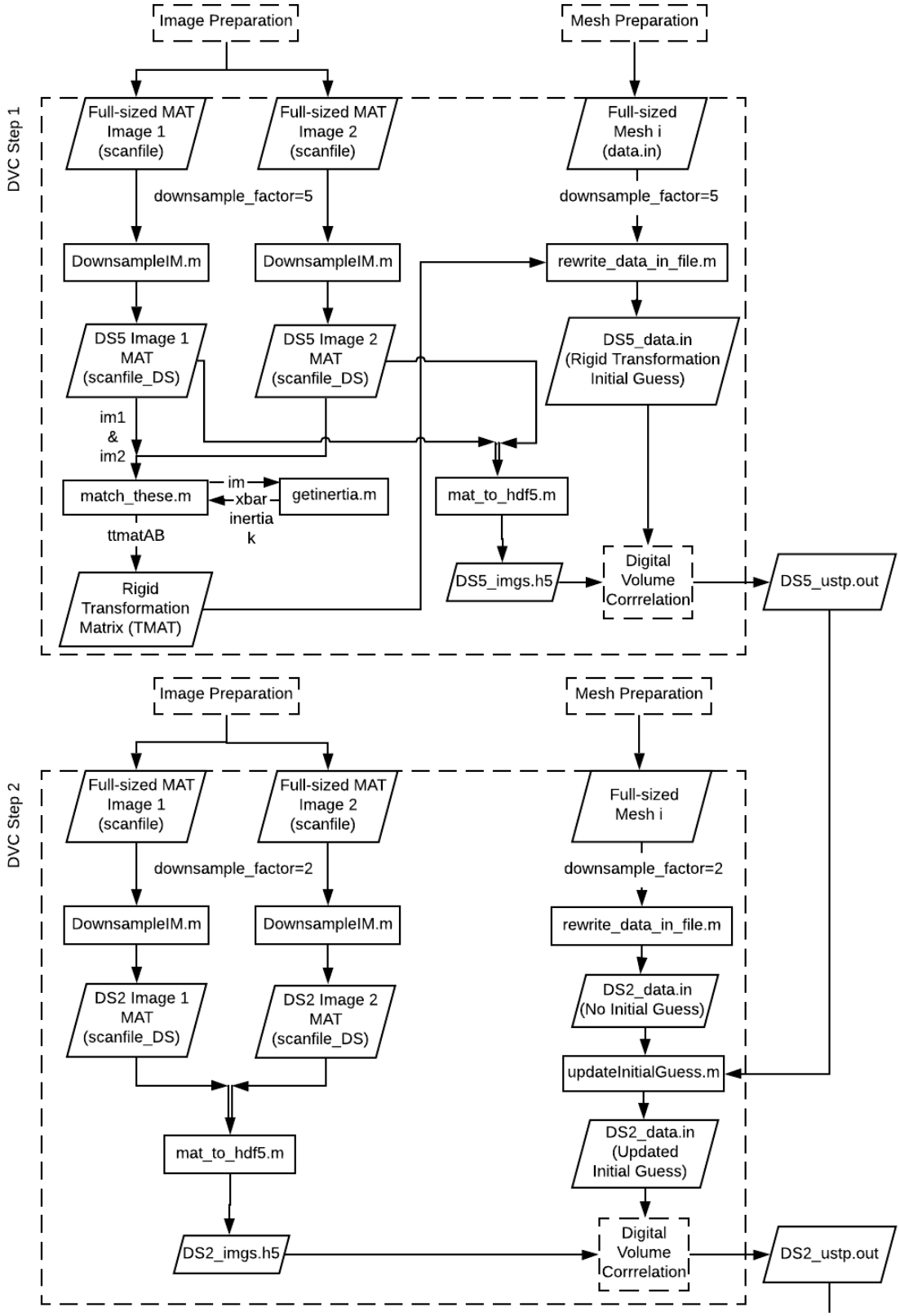


Figure 1.22: Overview flowchart of the DVC algorithm.


DVC Step 1
Image Preparation
Mesh Preparation
Full-sized MAT Image 1 (scanfile)
Full-sized MAT Image 2 (scanfile)
Full-sized Mesh i (data.in)
downsample_factor=5
downsample_factor=5
DownsampleIM.m
DownsampleIM.m
rewrite_data_in_file.m
DS5 Image 1 MAT (scanfile_DS)
DS5 Image 2 MAT (scanfile_DS)
DS5_data.in (Rigid Transformation Initial Guess)
im1 & im2
match_these.m
im
xbar
inertia
k
getinertia.m
mat_to_hdf5.m
ttmatAB
Rigid Transformation Matrix (TMAT)
DS5_imgs.h5
Digital Volume Corrrelation
DS5_ustp.out
DVC Step 2
Image Preparation
Mesh Preparation
Full-sized MAT Image 1 (scanfile)
Full-sized MAT Image 2 (scanfile)
Full-sized Mesh i
downsample_factor=2
downsample_factor=2
DownsampleIM.m
DownsampleIM.m
rewrite_data_in_file.m
DS2 Image 1 MAT (scanfile_DS)
DS2 Image 2 MAT (scanfile_DS)
DS2_data.in (No Initial Guess)
updateInitialGuess.m
mat_to_hdf5.m
DS2_data.in (Updated Initial Guess)
DS2_imgs.h5
Digital Volume Corrrelation
DS2_ustp.out

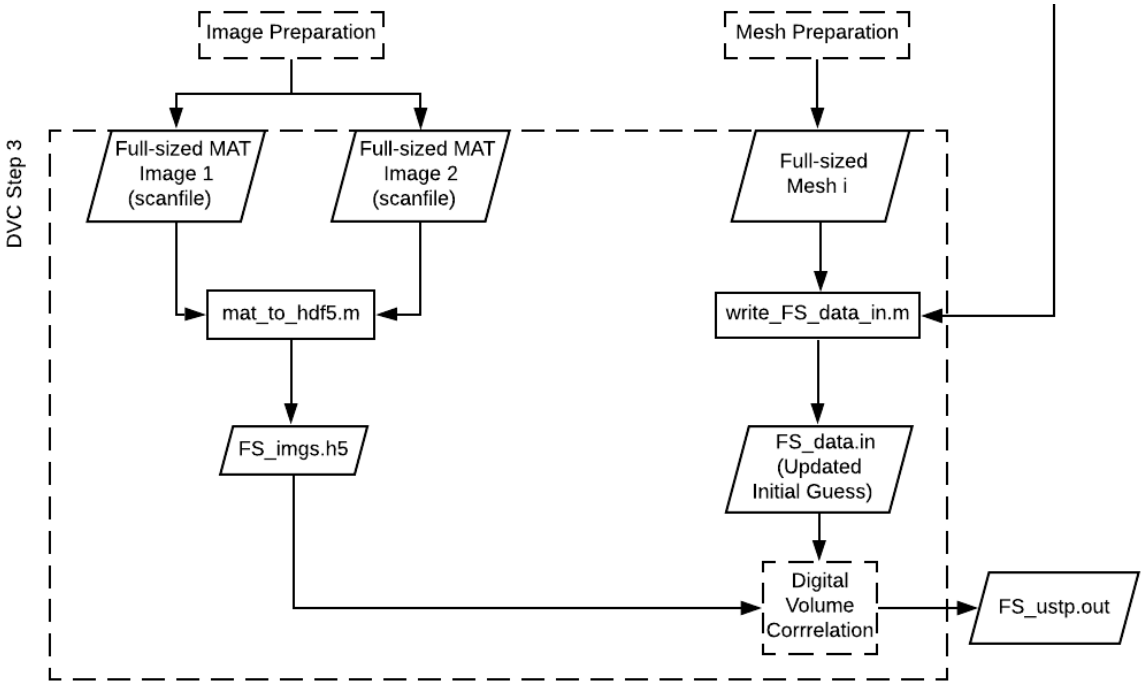


Figure 1.23: Detailed flowchart of the DVC algorithm.

2. IMAGE ACQUISITION

2.1. Optimizing scan and reconstruction parameters to facilitate proper DVC performance

2.1.1. Scan parameters

The primary parameters while setting up a scan are:

1) sample positioning relative to x-ray source and detector,
2) current and voltage,
3) exposure time, and
4) filter type.

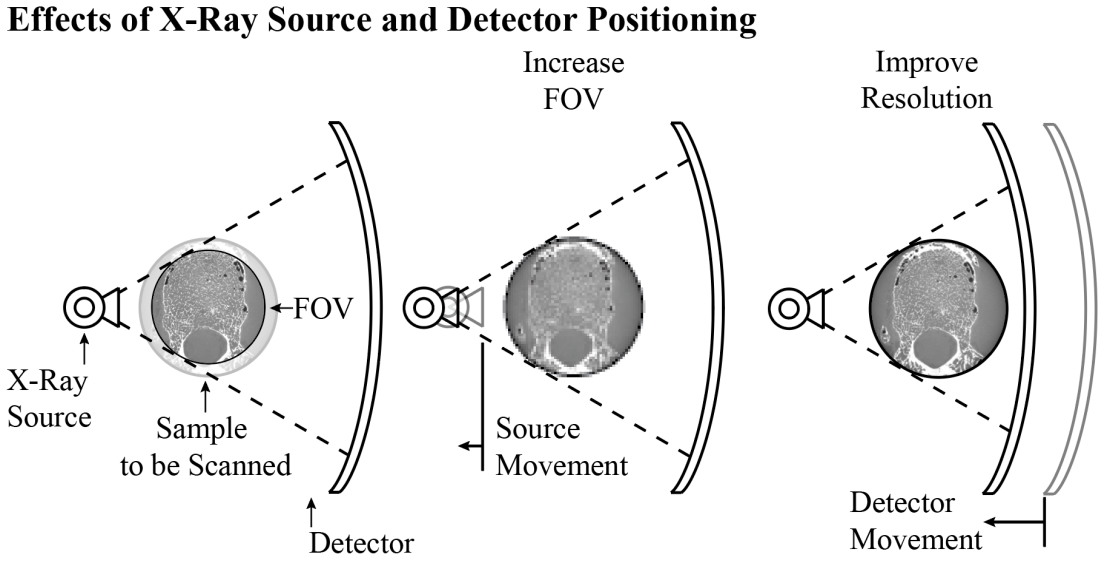


Figure 2.1: Schematic to show how x-ray source and detector positioning impact image field of view and resolution. The distance between the source and sample dictates the field of view (FOV). Distance between the source and detector dictates image resolution (voxel size).

We will briefly review how each of these parameters affect the final image and what to prioritize to optimize DVC performance. First, sample positioning relative to the x-ray source and detector determines the field of view (FOV), i.e. what the scan captures, and

image resolution, i.e. voxel size. Figure 2.1 demonstrates that the distance between the sample and source establishes the FOV and the distance between the source and detector determines the resolution. Typically, we first establish an appropriate FOV with source positioning and then optimize resolution with detector positioning.

The remaining three parameters influence the intensity range (brightness) of the image and are typically dependent on the type of material we scan. Current and voltage control the concentration of photons, while exposure time controls how long the aperture is open. To protect specimen that require a high concentration of photons in a scan, we use an appropriate filter to mitigate the damaging effects. Through a combination of these parameters, we can assure that we have large intensity for the feature we wish to track with DVC.

Now, if we think of these images as a collection of peaks we would like to align using a DVC algorithm, we recall from the 1D examples in section 1.2 that we ideally want our "feature" peaks to be distinguishable from the remaining material. For instance, with vertebrae images, we want the trabecular bone intensity to be distinguishable from the surrounding marrow. We promote this difference by having clear contrast. That is, high intensity values for bone and low intensity values for everything else. In addition, a high image resolution would widen the feature peaks, which further promotes convergence to an accurate displacement field. With trabecular bone distinguished through high intensity values that are spread over multiple voxels, we can more easily track trabecular bone movement during deformation of the vertebra.

2.1.2. Reconstruction parameters

The raw image introduces features in the image that are not present in the physical sample. These imperfections caused directly by the scanner are called “image artifacts”. To reduce them, we reconstruct an image applying corrective processing steps. Some of these steps include:

1) Center shifting
2) Beam hardening
3) Rotation angle
4) Smoothing
5) Byte Scaling

Without delving too much into detail on the expansiveness of image reconstruction, we will briefly cover the processes that a typical user should be actively aware of when preparing an image. With this criterion in mind, we exclude center shifting and beam hardening from discussion as these are steps typically addressed by automated algorithms in reconstruction software.

Rotation angle defines the angle of the sample prior to reconstruction. With the exception of some minor interpolation, the rotation angle has little effect on the intensity distribution.

Smoothing is as straightforward as one would expect: we smooth out the intensity distribution to reduce “noisy” peaks present in the image. We must avoid over-smoothing as this may smooth over features that are physically present in the sample.

Byte scaling is a process used when converting the intensity values from FLOAT to UINT16. In this step, we define an intensity range (FLOAT), which is then scaled to the full range of the UINT16 class (1 to 2^{16}). This step directly affects image contrast.

Regardless of what reconstruction parameters we use, it is critical that we use the same settings for each image pair we will match with DVC.

2.2.Sources of reproducibility error

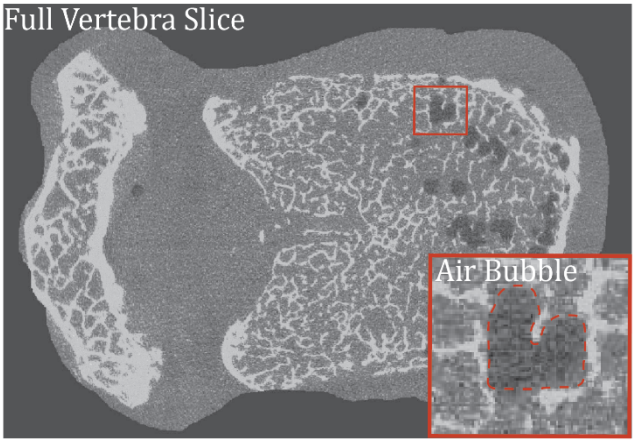


Figure 2.2: Zoomed in image of an air bubble in a µCT scan of a human vertebra. Notice the darker intensity pattern.

Unfortunately, even if we maintain identical scan and reconstruction parameters, there will remain unwanted image differences caused by both the scanner and biological sample degradation. We define these as *repeatability artifacts*. For example, the scanner introduces noise in its purest definition and micro-motion of the sample. To explain the second, after removal and replacement between scans, the scanner introduces sub-voxel, rigid rotation. The slight motions of the base, which supports the sample during the scan, causes this micro-motion. Finally, biological sample degradation introduces air bubbles in subsequent images that will grow in size the longer the biological sample is at room temperature or heated up, which is typical in μCT scans. When our goal is to track a displacement field between a pair of images, these air bubbles are not image differences we wish to track. Nevertheless,

this image characteristic causes the displacement field, calculated by DVC, to slightly deviate from the truth.

3. DVC DATA PREPARATION

3.1. Motivation

Our DVC algorithm has two file inputs: an HDF5 file containing both images to be matched (imgs.h5) and a data input file (data.in). The first is prepared by isolating the region of interest (ROI) from the images. This is typically done through “contouring”, which is a segmentation process where a user manually draws outlines the ROI slice-by-slice. The second file contains necessary data: various DVC parameters (such as regularization parameter), image details (size, voxel side length, and location of first non-zero slice in each plane), mesh information (element type, node quantity and location, element quantity and connectivity), and initial guess of nodal displacement. Preparing these two input files is non-trivial and therefore requires explanation.

3.2. Image Preparation

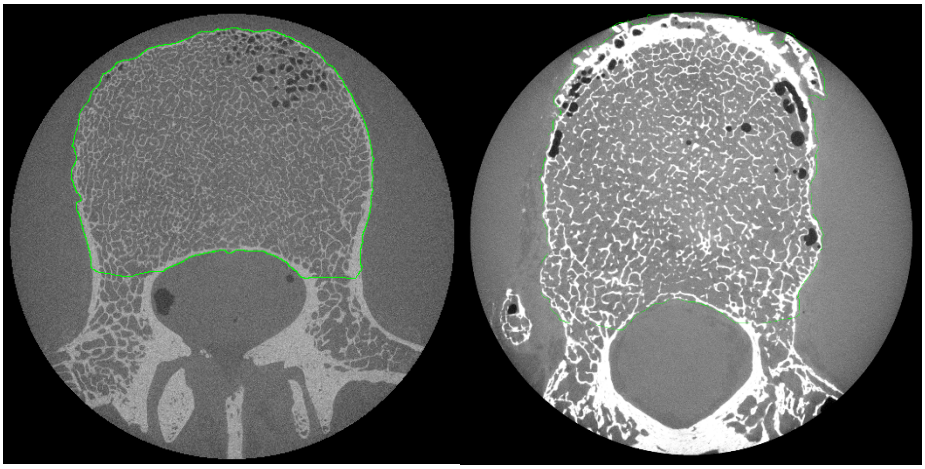


Figure 3.1: Sample contouring performed on Scanco (left) and Fiji (right), which we use to create the surface (STL) to be meshed and the images to be registered, respectively.

3.2.1. Contour

Contouring is the process used to isolate the region of interest (ROI). The contour defines a region outside of which we zero all intensity values. This is called "masking" an image. We have used two software packages: Scanco and Fiji (v1.52p) to contour our images (see Figure 3.1).

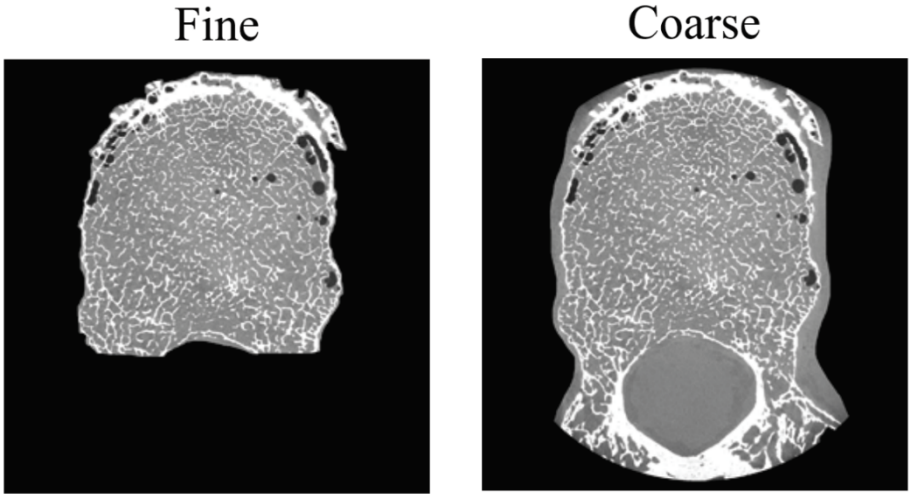


Figure 3.2: Fine and coarse contours (of a single slice) applied to create a mask of the same vertebral image.

We use two categories of contours: a fine and coarse contour (see Figure 3.2). The fine contour is only created for the undeformed image. As its name suggests, the fine contour closely outlines the edges of the vertebral body. We use it to define the surface file used to create the finite element mesh in section 3.3. A coarse contour is created for both the undeformed and deformed images. The goal of a coarse contour is to isolate the bone of interest from adjacent ones. With our vertebrae, we seek to eliminate the adjacent vertebrae from the images. This step helps us avoid falsely matching bone intensity patterns that are not of interest during the rigid registration step explained in section 4.3.

3.2.2. Threshold

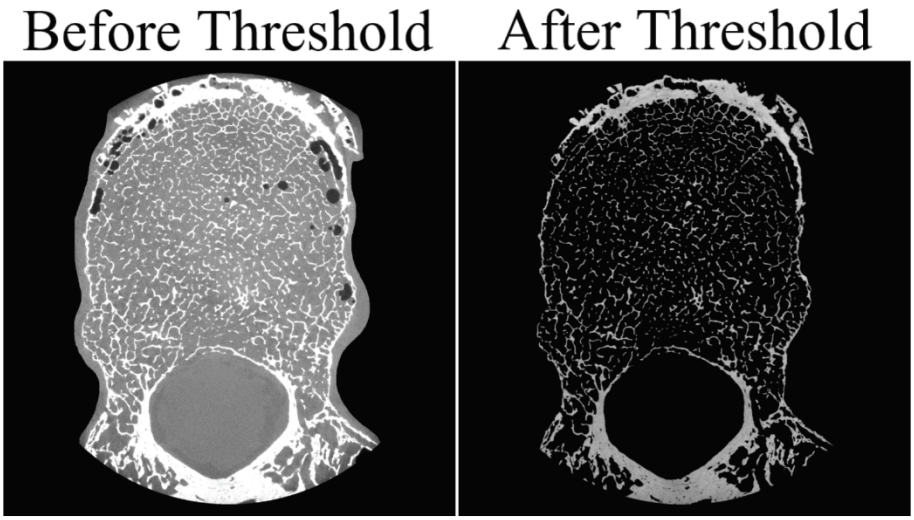


Figure 3.3: Threshold applied to a vertebral image.

Thresholding is the process of zeroing image intensities that are below a set value (Figure 3.3). We do so to isolate the intensity range of interest. When studying vertebral fracture, we isolate the intensity values of bone. When using our rigid registration algorithm presented in section 4.3, we threshold our images to promote rigid alignment of the

trabecular bone. However, in our 3D, DVC algorithm, we do not threshold the images. Instead, we only use the coarsely masked image shown in Figure 3.2.

3.2.3. Image Data Format

Regardless of the scanner used to acquire our μCT images, our goal is to import all images to a MAT format: the native MATLAB variable storage format. This enables us to perform image-processing steps of downsampling (see section 3.5), rigid registration (see chapter 4) and exportation to HDF5 format.

HDF5, or Hierarchal Data Format (version 5), is an open-source, binary storage format that is efficient in handling large data such as the μCT 3D, image matrices. With this format, we are able to store both images in a single file (imgs.h5), which is an input read in the DVC Fortran code.

3.3. Finite Element Mesh

3.3.1. Motivation

We introduced the benefits of local spatial averaging with 1D DVC in chapter 1. In our 3D DVC approach, we subdivide via finite elements (FE) that conform to the geometry of the region of interest. We can compute finite-element based strain using basic structural mechanics equations.

In the following subsections, we will go over the process of creating a surface, a finite element mesh, and finally, a data input file for DVC.

3.3.2. Creation of a surface from a 3D Image (Scanco and Fiji)

From the contour (see section 3.2.1), we create a surface (STL file) which provides the ROI geometry used for the development of a finite element mesh. Regardless of which software is used to create the STL, the basic image processing steps prior to doing so are the same:

1) Convert the grayscale image to a binary mask.
2) Smooth out the edges of the ROI.
3) Downsample the binary images.

Conversion from grayscale to a binary mask is standard for any software that produces an STL from an image stack. Edge smoothing helps avoid the production of invalid elements. Finally, downsampling minimizes the computational cost of creating an STL file that is easier to open and work with in any mesh-creating software.

3.3.3. Mesh Creation (IA-FEMesh)

Our DVC algorithm is designed to use hexahedral, finite elements (see Figure 3.4). We use a meshing software, Iowa Finite Element Mesh (IA-FEMesh), to produce the mesh under user guidance.

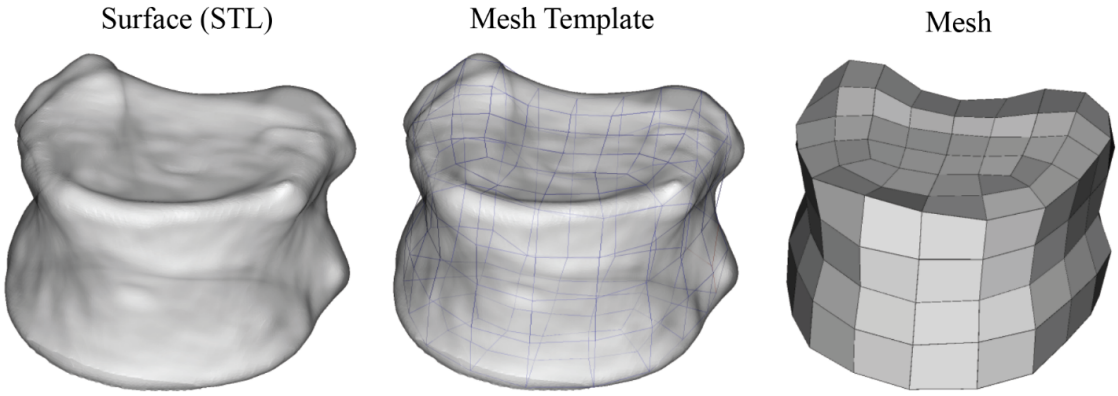


Figure 3.4: Sample mesh created using the IA-FEMesh software. We start with a surface file, follow with a manually created mesh template (or block structure), and finish with creating the mesh.

To compute strain for any mesh element given the node displacements, one must assume that the element encloses only one material. However, in images of bone a single mesh element typically encloses two materials: trabecular bone and marrow. If our element volume is sufficiently large ($125mm^3$ for vertebrae), then we can assume that everything enclosed by the element represents a single material. This is the continuum assumption.

The critical outputs we seek from any meshing software are node coordinates and element connectivity. The nodal coordinates are given in physical space. Because the DVC Fortran code is written to import nodal coordinates mapped in image space, we need to convert nodal coordinates from physical to image space for the data input file.

3.4. Correspondence between Physical and Image Space

3.4.1. Indexing in Physical and Image Space

At this point, it is critical that we understand how to navigate matrix and image indexing conventions. First, we look at how we index a typical 2D matrix. To do so, let's look at the following 3D matrix:

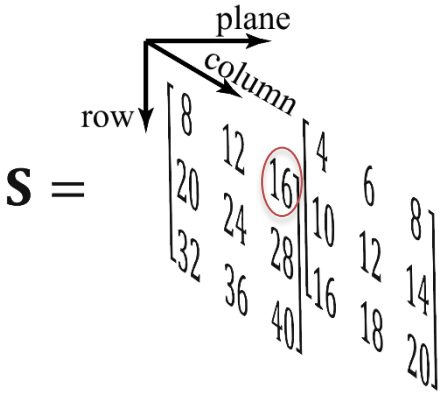


Figure 3.5: A 3x3x2 matrix, S, is shown as an example to understand indexing. The arrows indicate the direction in which indices increase in value.

A typical 3D matrix indexing system follows the convention: (row, column, plane). For example, we extract the value "16" from **S** in Figure 3.5 with the index (1, 3, 1); that is row 1, column 3, and plane 1.

In our work, an image is also a matrix whose values (grayscale intensities) are dictated by the density of the scanned object. Image indexing is performed in physical space, which follows a right-handed coordinate system of (x, y, z), or (column, row, slice) in matrix terms, with the z-axis pointing into the page (Figure 3.6 shows this convention in 2D). With the z-axis pointing into the page and considering right-handedness, the y-axis must point down. Using this convention, we extract "16" from **S** in Figure 3.5 with the index (3, 1, 1); that is column 3, row 1, and slice 1. We will use an

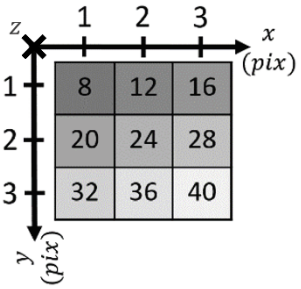


Figure 3.6: A 3x3 image where the color of a single pixel reflects the value inside (higher pixel values are brighter). Note that we are using a right-handed coordinate system, with the z-axis pointing into the page. The tick spacing seen in the coordinate system shown is equal to one pixel side length. Note that the origin is not located at the top left corner of the image. Instead, the origin is offset by a half-pixel side length away in both x and y from the top left corner of the image.

image indexing convention in our derivations moving forward when accessing the intensity values of any given image. However, we must also keep in mind that to access a given component of an image matrix in any coding language, we must do so with a matrix indexing convention.

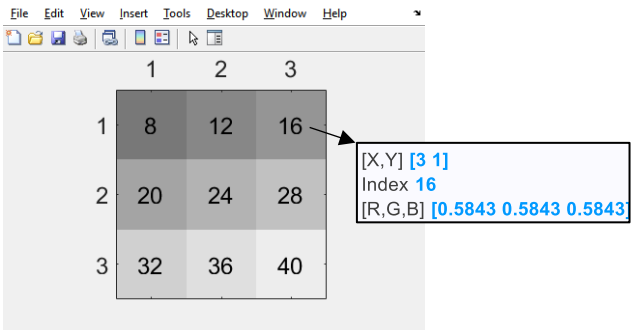


Figure 3.7: The first plane/slice of matrix *S* displayed with the `imagesc` MATLAB function. Note that when using the "data tips" tool in the figure window, the voxel coordinates, index value, and [R,G,B] of the selected pixel are displayed. This tool shows that the coordinates of this pixel are (3,1), which follows the image indexing system above. However, when accessing the pixel with the value "16" from the MATLAB command window, we type in "`S(1,3)`". Thus, MATLAB extracts matrix values with a typical matrix indexing convention.

Now, when using MATLAB, we come across situations where both indexing conventions, matrix and image, are used. For example, when displaying an image with functions such as `imshow` or `imagesc`, the indexing scheme is the physical system seen in Figure 3.6 (sample MATLAB figure window seen in Figure 3.7). However, if we wish to extract a matrix element in code, then we use a typical matrix indexing convention.

Finally, when transforming images, we must know the origin about which said transformation is applied and understand how to locate pixels in a continuous image coordinate system. The convention used here is that the coordinate of any given pixel is located at its center. Also, if we assume the axes are in units of pixels, then the origin is located near the top left corner of the matrix and offset by a half-pixel side length in x and y from the top left corner of the first pixel (Figure 3.6).

3.4.2. Physical Coordinate System to Image Coordinate System

Once we import the mesh node coordinates, we must calculate their placement on the image itself. The most relevant questions when doing so are:

1) Where is the physical coordinate system origin located relative to the image coordinate system?
2) What is the image resolution (conversion between a voxel side-length and the physical length it represents)?

To answer the first, the STL's origin location is either at the center of the image (Scanco produced STLs) or coincident with the origin of the image coordinate system (Fiji produced STLs). The answer to the second question is simply dependent on the scan settings used. To address both questions more explicitly, we map the nodal coordinates provided in a physical coordinate system by the meshing software ($\bar{x}_{mm}$ in units of mm), to an image coordinate system ($\bar{x}_{vox}$ in units of vox) expected in the DVC Fortran code with the following relationship:

$$\bar{x}_{vox} = \frac{\bar{x}_{mm}}{r} - \bar{x}_0 \tag{3.1}$$

where r represents the image resolution (in units of mm/vox) and $\bar{x}_0$ represents the coordinates of the STL origin location (in units of vox). As image coordinates must be integer values, we typically round out any decimals.

3.4.3. Conversion of Nodal Coordinates and Element Connectivity for DVC Input File

In addition to converting node coordinates to image space, we reorder the nodes listed in the nodal connectivity matrix. This simple procedure is depicted in Figure 3.8.

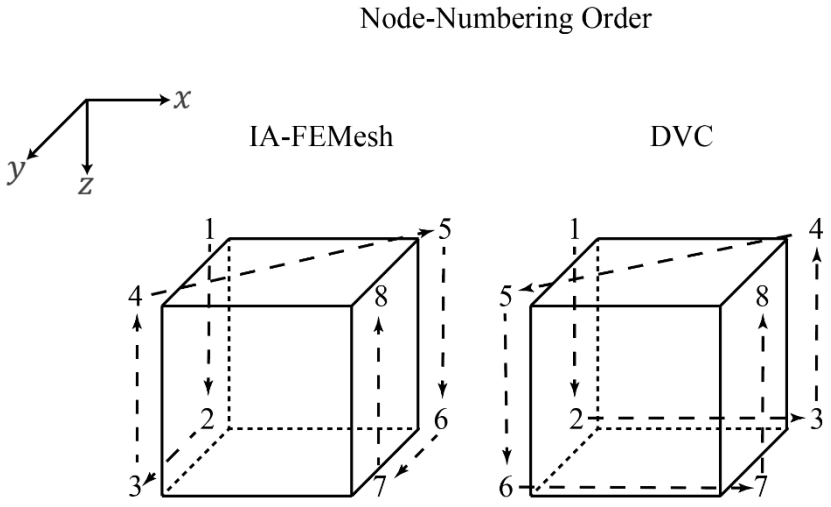


Figure 3.8: Schematic to demonstrate how node-numbering order differs between IA-FEMesh and the DVC Fortran code.

3.5. Image Downsampling

3.5.1. Motivation

DVC convergence speed is dependent on iteration speed and quantity. Iteration speed is determined by the sheer size of the image matrices that must be analyzed per iteration. Iteration quantity depends on the accuracy of the initial guess of the displacement field. Image downsampling addresses both: matrix sizes become much smaller and the outcome displacement field can be scaled back up to full-size to provide an accurate initial guess for the full-sized image.

In our data processing flow, we downsample the images twice: by scaling factors of five and two. The DVC results from the factor of five are used to produce the displacement

initial guess for the factor of two. The output displacement field from the factor of two is then used as the initial guess for the full-sized image pair. This pyramidal process improves convergence speed.

In addition to improving DVC convergence speed, downsampling serves as a low-pass filter to help reduce the possibility of convergence to a local minimum (as introduced in section 1.2.7).

3.5.2. Image Resampling (`imresize3`)

To reduce our image size, we resample it by the desired scaling factor. We use the `imresize3` built-in MATLAB function available in the Image Processing Toolbox. The primary benefits of this function are its computational efficiency and availability of high-order interpolation kernels. We use the default kernel, a cubic kernel, in `imresize3`. As compared to lower-order kernels, the cubic kernel produces smoother images because it uses a larger neighborhood of voxels to compute each new intensity. One benefit of this increased smoothness is that the locations of features (peaks in intensity) in the image are better preserved, which leads to more accurate registration of the downsampled images in the next step of the DVC workflow.

When using `imresize3`, we use the default options: antialiasing and cubic kernel. Briefly, antialiasing is a technique used to minimize distortion artifacts (aliasing) when downsampling an image, and cubic interpolation is a method of interpolation that uses a weighted sum of pixels in the nearest 4-by-4-by-4 neighborhood.

Image resampling with cubic interpolation and antialiasing is performed by:

1. Converting a discrete-domain image matrix, I_{xyz}, to a continuous-domain image function, $I(x,y,z)$, using a cubic kernel, $k_{Keys}(h)$, which interpolates between matrix components.
2. Applying a low-pass filter to $I(x,y,z)$, by scaling $k_{Keys}(h)$.
3. Converting $I(x,y,z)$, to a downsampled, discrete-domain image matrix, $\left[\mathrm{I_{xyz}}\right]_{\mathrm{DS}}$, *by resampling* $I(x,y,z)$ at the desired rate, s: $\left[\mathrm{I_{xyz}}\right]_{\mathrm{DS}} = I(sx,sy,sz)$. In our case, we use $s = 2$ or 5.

This entire process is accomplished with the following function:

$$\left[\mathrm{I_{xyz}}\right]_{\mathrm{DS}} = \sum_{\mathrm{l=a}}^{\mathrm{b}} \sum_{\mathrm{m=a}}^{\mathrm{b}} \sum_{\mathrm{n=a}}^{\mathrm{b}} \mathrm{k(l)k(m)k(n)}\, I_{sx,sy,sz} \tag{3.2}$$

In equation (3.2), the summation limits, when using a cubic interpolation kernel, start at $a = -2$ and $b = 1$ to define the nearest 4-by-4-by-4 neighborhood. When downsampling, summation limits spread out as a function of the sampling rate. The cubic interpolation kernel is presented by Keys 1981[16]:

$$\mathrm{k_{Keys}(h)} = \begin{cases} \frac{3}{2}|\mathrm{h}|^3 - \frac{5}{2}|\mathrm{h}|^2 + 1 & ;\text{if } 0 \le |\mathrm{h}| < 1 \\ -\frac{1}{2}|\mathrm{h}|^3 + \frac{5}{2}|\mathrm{h}|^2 - 4|\mathrm{h}| + 2 & ;\text{if } 1 \le |\mathrm{h}| < 2 \\ 0 & ;\text{if } |\mathrm{h}| \ge 2 \end{cases} \tag{3.3}$$

This kernel is scaled to apply the low-pass filter needed for antialiasing:

$$\mathrm{k(h)} = \frac{1}{\mathrm{s}} * \mathrm{k_{Keys}}\left(\frac{\mathrm{h}}{\mathrm{s}}\right) \tag{3.4}$$

Equation (3.4) provides the kernel used in (3.2). Further detail on `imresize3` may be found in the following online MathWorks pages:

https://www.mathworks.com/help/images/resize-an-image.html

https://blogs.mathworks.com/steve/2017/01/16/aliasing-and-image-resizing-part-3/

https://www.mathworks.com/help/images/ref/imresize3.html

3.5.3. Implementation

We created a MATLAB function that produces and saves a downsampled version of the input image using `imresize3`. Our function, `downsampleIM.m` outputs a downsampled image (`scanfile_DS`) and receives, as inputs, the image to be scaled (`image`), the downsampling factor (`down_sample_factor`), and filename (`name_of_image`) and directory (`image_path`) of the output image. The downsampled images (`scanfile_DS`) are used as the inputs for the function `match_these.m` included below in section (4.3.4).

```
%-------------------------------------------------------------------------%
%                                                                         %
% MATLAB function: downsampleIM.m
%                                                          version 1.1    %
%-------------------------------------------------------------------------%
% Author        : Johnfredy Loaiza
% Version       : 1.1
% LastEdit date : 14 February 2020
% Description   : produce and save a downsampled version of the input
%                 images
% 14 Feb 2020   : Johnfredy Loaiza: Creation
% 3 Jun 2020    : Johnfredy Loaiza: Added comments for the DVC tutorial
%
%-------------------------------------------------------------------------%
%                                                                         %
%               JL: Boston University                                     %
%               Boston, MA                                                %
%                                                                         %
%-------------------------------------------------------------------------%
%
%-------------------------------------------------------------------------%
%               Components/Includes                                       %
%-------------------------------------------------------------------------%
%
% none
%
%-------------------------------------------------------------------------%
%  Input(s)    : image = 3D matrix of image intensities
%                down_sample_factor = downsampling factor so that final
%                                     image will have a size of
%                                     originalSize/down_sample_factor
%                image_path_name = string that indicates location to save
%                                  the downsampled image
%                name_of_image = name of image file to be saved
%  Output(s)   : scanfile_DS = downsampled 3D matrix of image intensities
%-------------------------------------------------------------------------%

function                                scanfile_DS                                =
downsampleIM(image,down_sample_factor,image_path,name_of_image)
%% Define new image name
    new_image_name = ...

[image_path'/'erase(name_of_image,'.mat')'_DS'num2str(down_sample_factor)];

% Determine if downsampled image already exists
    if isfile(new_image_name)==0
        %% Size of element (must be cubic)
        scale = 1/down_sample_factor;

        %% Downsample image
        scanfile_DS = imresize3(image,scale);

        %% Save the matrix
        save(new_image_name,'scanfile_DS','-v7.3');
    else
        load(new_image_name,'scanfile_DS');
    end
end
```

4. RIGID REGISTRATION WITH MOMENTS OF MASS

To calculate our first initial guess (for the images downsampled by a factor of five above) we compute a rigid transformation matrix. This chapter covers how we estimate the rigid movement of a body from images of the body before and after it is displaced. To estimate this mapping, we perform a rigid registration, or alignment, of those images, which produces the transformation matrix that maps the body position from one image to the other.

Though registration is offered as an automated process in some image processing software (Scanco IPL, ImageJ/Fiji, MATLAB), explicit explanation is often omitted or unknown to the user. We describe exactly the practice we are using as part of the DVC data processing flow.

4.1. Transformation Matrix for Mapping Rigid Body Motion

4.1.1. Glossary of Mathematical Terms

Mathematical terms are presented in order of appearance.

$[\mathbf{v}]_2$: two-component vector with components v_x and v_y

$[\mathbf{R}]_2$: 2x2 rotation matrix

x_C, y_C: coordinates of a point about which a rotation is applied to a vector or body

$[\mathbf{t}]_2$: two-component vector representing a translation vector with components t_x and t_y

$[\mathbf{R}]_3$: 3x3 rotation matrix

$[\mathbf{t}]_{\mathbf{3}}$: three-component translation vector with components t_x, t_y, and t_z

$[\mathbf{R}]_{\mathbf{4}}$*:* 4x4 transformation matrix of pure rotation

$[\mathbf{T}]_{\mathbf{4}}$*:* 4x4 transformation matrix of pure translation

RT*:* 4x4 rigid transformation matrix

$[\mathbf{RT}]_{\mathbf{z}}$: 4x4 rigid transformation specific to a rotation about the z-axis

$[\mathbf{v_1}]_{\mathbf{2}}$: two-component, body vector that starts at a central point of a rectangular body and tangentially points at one rectangle side

$[\mathbf{v_2}]_{\mathbf{2}}$: two-component body vector that starts at a central point of a rectangular body and tangentially points at another rectangle side. This vector is perpendicular to $[\mathbf{v_1}]_{\mathbf{2}}$.

$\bar{x}, \bar{y}, \bar{z}$: mass centroid coordinates of a body (including cuboid)

$\rho(x, y, z)$*:* mass density function of a body

M: total mass, or zeroth moment of mass, of a body

M_x, M_y, M_z: first moments of mass of a body about each axis

Inertia*:* inertia matrix derived from the second moment of mass about the mass centroid with components $Inertia_{xx}$, $Inertia_{yy}$, $Inertia_{zz}$, $Inertia_{xy}$, $Inertia_{xz}$, and $Inertia_{yz}$.

u, **v**, **w**: eigenvectors of the inertia matrix

$\mathbf{R_A}$, $\mathbf{R_B}$*:* principal inertia matrices composed of the inertia matrix eigenvectors

$[\mathbf{R_A}]_\mathbf{z}$, $[\mathbf{R_B}]_\mathbf{z}$: 3x3 principal inertia matrices for the 2D configurations illustrated in this chapter

$\mathbf{R_{180° \text{ about } x}}$, $\mathbf{R_{180° \text{ about } y}}$, $\mathbf{R_{180° \text{ about } z}}$: test matrices to determine proper orientation of principal inertia matrices

T: 3x3x3 third moment of mass matrix with component T_{ijk}

k: third moment vector of a body about the mass centroid

d: four-component vector with components $d_1 - d_4$ used to determine correct principal inertia matrix orientation

$\mathbf{R_{correct}}$: rotation matrix that orients the principal inertia matrices correctly

$\mathbf{R_{Bcorrect}}$: corrected principal inertia matrix

$\mathbf{R_{AB}}$: rotation matrix orienting one body configuration to the other

S: sample 3x3 matrix

lx, ly, lz: side lengths of cuboid

$\Delta x, \Delta y, \Delta z$: voxel-side lengths

nx, ny, nz: number of voxel along each axis direction

x_i, y_j, z_k: indexing/sampling variables

M_I: total image "mass"

I_{ijk}: 3d image matrix with grayscale intensity values in each component

$\bar{x}_I, \bar{y}_I, \bar{z}_I$: mass centroid coordinates of image

$\mathbf{Inertia_I}$ *:* image inertia matrix about the mass centroid

r^2*:* distance from mass centroid to any point on cuboid

$\mathbf{k_I}$*:* third moment vector of image

r_I^2*:* distance from image mass centroid to any voxel in image

4.1.2. Rotation and Translation of a Vector

Rotation of a two-component vector, $[\mathbf{v}]_\mathbf{2} = [v_x\ v_y]$, is accomplished through a product of that vector and a 2x2 rotation matrix, $[\mathbf{R}]_\mathbf{2}$:

$$[\mathbf{v}']_\mathbf{2} = [\mathbf{v}]_\mathbf{2}[\mathbf{R}]_\mathbf{2} \tag{4.1}$$

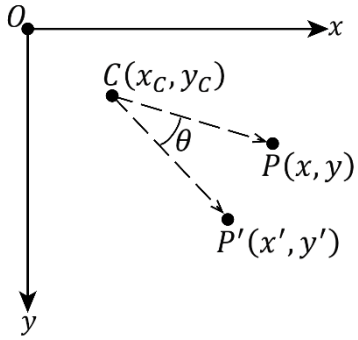


Figure 4.1: Depiction of moving a point, P by rotating the vector $\overrightarrow{\mathbf{CP}}$ about point C. By allowing C to be placed anywhere, this method allows us to map P to any location in the coordinate system. Though not depicted, the z-axis in the right-handed coordinate system above goes into the page. Also, note that the direction of rotation follows the positive direction (about the z-axis).

All right-handed rotation matrices have the property that $\mathbf{R}^{-1} = \mathbf{R}^{\mathrm{T}}$, and $|\mathbf{R}| = 1$. We begin with the example depicted in Figure 4.1 (left). In this example, we wish to map the point P with coordinates (x, y) to a new position, P' with coordinates (x', y'), by rotating the vector that has its "tail" at point C and its "head" at point P. Point C with coordinates (x_C, y_C) is the point about which the rotation of the vector will be performed. A positive rotation follows the right-handed rule where the z-axis, though not depicted above, goes into the page. In Figure 4.1, we rotate about this z-axis. We define each term in equation (4.1) as so:

$$[\mathbf{v}']_\mathbf{2} = [x' - x_C \quad y' - y_C] \tag{4.2}$$

$$[\mathbf{v}]_\mathbf{2} = [\mathrm{x} - \mathrm{x_C} \quad \mathrm{y} - \mathrm{y_C}] \tag{4.3}$$

$$[\mathbf{R}]_2 = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix} \tag{4.4}$$

Next, by substituting equations (4.2) and (4.3) into (4.1), we get:

$$[x' - x_C \quad y' - y_C] = [x - x_C \quad y - y_C][\mathbf{R}]_2 \tag{4.5}$$

We rearrange equation (4.5):

$$[x' \; y'] = [x \; y][\mathbf{R}]_2 + [x_C \; y_C] - [x_C \; y_C][\mathbf{R}]_2 \tag{4.6}$$

From equation (4.6), we can describe the mapping of the point located at (x, y) in two ways (see Figure 4.2): 1) a pure rotation about some point C or 2) a rotation about the origin plus two additional terms.

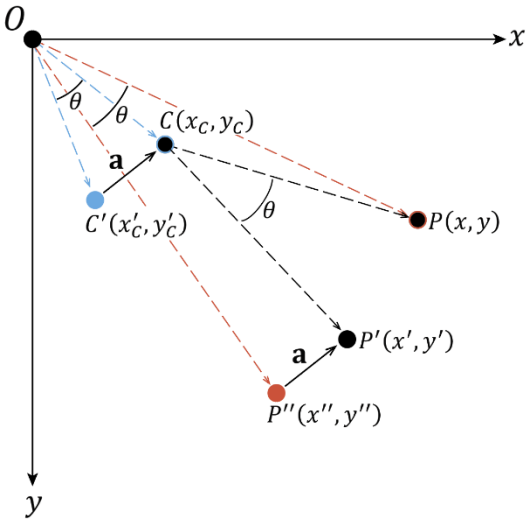


Figure 4.2: Rotation of the vector $\overrightarrow{CP}$ about C by θ to map P to P′ (black) is equivalent to rotating the vector $\overrightarrow{OP}$ by that same θ, which produces P″ (red) and offsetting it. The offset vector, a, is the same as the vector from C and its post-rotated, by θ, counterpart C′ (blue); i.e. $\mathbf{a} = [x_C \; y_C] - [x_C \; y_C][\mathbf{R}]_2$.

Let us define those two terms as:

$$\left[\mathrm{t_x}\ \mathrm{t_y}\right] = [x_C\ y_C] - [x_C\ y_C][\mathbf{R}]_\mathbf{2} \tag{4.7}$$

Applying this definition, equation (4.6) becomes:

$$[x'\ y'] = [x\ y][\mathbf{R}]_\mathbf{2}\ + [t_x\ t_y] \tag{4.8}$$

$[\mathbf{t}]_\mathbf{2} = [t_x\ t_y]$ is known as the translation vector and represents the correction that needs to be added in order to relate rotation about the origin to rotation about a different point, C. Stated differently, the translation vector depends on the point of rotation.

Equation (4.8) demonstrates a two-step, mapping process where we first rotate then translate. Now, let us see how this equation changes for two other mapping methods. In the case where we want to map point *P* with only translation, i.e. no rotation is present, $[R]$ is an identity matrix and equation (4.8) becomes:

$$[\mathrm{x}'\ \mathrm{y}'] = [\mathrm{x\ y}] + [\mathbf{t}]_\mathbf{2} \tag{4.9}$$

where P is simply shifted. Alternatively, when the vector tail is located at the origin, as in Figure 4.1 (right), equation (4.8) becomes a pure rotation about the origin:

$$[x'\ y'] = [x\ y][\mathbf{R}]_\mathbf{2} \tag{4.10}$$

In 3D, equations (4.8) and (4.7) become:

$$[x'y'z'] = [x\ y\ z][\mathbf{R}]_3 + [\mathbf{t}]_\mathbf{3} \tag{4.11}$$

$$[\mathbf{t}]_{\mathbf{3}} = [\mathrm{x_C}\ \mathrm{y_C}\ \mathrm{z_C}] - [\mathrm{x_C}\ \mathrm{y_C}\ \mathrm{z_C}][\mathbf{R}]_{\mathbf{3}} \tag{4.12}$$

where $[\mathbf{t}]_{\mathbf{3}} = [t_x\ t_y\ t_z]$ represents a three-component translation vector and $[\mathbf{R}]_{\mathbf{3}}$ a 3x3 rotation matrix:

$$[\mathbf{R}]_3 = \begin{bmatrix} \mathrm{R}_{11} & \mathrm{R}_{12} & \mathrm{R}_{13} \\ \mathrm{R}_{21} & \mathrm{R}_{22} & \mathrm{R}_{23} \\ \mathrm{R}_{31} & \mathrm{R}_{32} & \mathrm{R}_{33} \end{bmatrix} \tag{4.13}$$

4.1.3. Rigid Body Transformation Matrix

Let's define a body in space as a volume with arbitrary shape. Rigid motion of that body consists of both rotation and translation of that body as described in equation (4.11). We now show that (4.11) may alternatively be represented as a 4x4 transformation matrix.

The 4x4 transformation matrix of pure rotation is:

$$[\mathbf{R}]_{\mathbf{4}} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.14}$$

The 4x4 transformation matrix of pure translation is:

$$[\mathbf{T}]_{\mathbf{4}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix} \tag{4.15}$$

Both rotation and translation transformation matrices are compiled into a single rigid transformation matrix, **RT**, by multiplying equations (4.14) and (4.15):

$$\mathbf{RT} = [\mathbf{R}]_4[\mathbf{T}]_4 \tag{4.16}$$

From now on, all our transformation matrices will be 4x4, and therefore we drop the subscript 4 for clarity. Equation (4.16) expands to:

$$\mathbf{RT} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix} \tag{4.17}$$

This rigid transformation matrix, when applied to any vector pointing to a single point enclosed by the body, produces its new location. The points enclosed by the body in the initial configuration have the coordinates (x, y, z) and those enclosed in the new configuration have the coordinates (x', y', z'). The product to calculate the new coordinates is:

$$[x'y'z'1] = [x\ y\ z\ 1]\mathbf{RT} \tag{4.18}$$

When we expand equation (4.18) we find:

$$x' = x * R_{11} + y * R_{21} + z * R_{31} + t_x \tag{4.19a}$$

$$y' = x * R_{12} + y * R_{22} + z * R_{32} + t_y \tag{4.19b}$$

$$z' = x * R_{13} + y * R_{23} + z * R_{33} + t_z \tag{4.19c}$$

We see that equations (4.19a–c) equal equation (4.11) in an expanded form, as we should expect.

As an example, let us look at how the transformation matrix is used in a 2-D case to map a single (material) point on a body in one configuration to another.

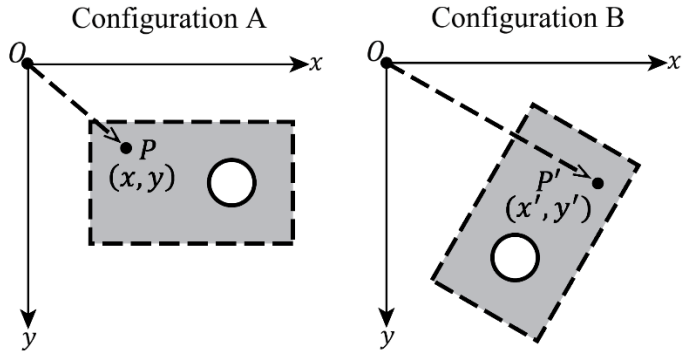


Figure 4.3: A sample body with an initial configuration (A) is rigidly moved to a new configuration (B) so that the point P with initial coordinates (x, y) has new coordinates (x', y').

In the 2-D case shown in Figure 4.3, the rotation matrix in **RT** becomes a rotation about the z-axis and the translation in the z-direction is zero:

$$[\mathbf{RT}]_{\mathbf{z}} = \begin{bmatrix} cos(\theta) & sin(\theta) & 0 & 0 \\ -sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & 0 & 1 \end{bmatrix} \tag{4.20}$$

We calculate the new coordinates (x', y') with the product:

$$[x'y'0\ 1] = [x\ y\ 0\ 1][\mathbf{RT}]_{\mathbf{z}} \tag{4.21}$$

If we expand equation (4.21) to calculate x' and y' individually, we get:

$$x' = x * cos(\theta) - y * sin(\theta) + t_x \tag{4.22a}$$

$$y' = x * sin(\theta) + y * cos(\theta) + t_y \tag{4.22b}$$

The previously mentioned two-step process appears once again in equations (4.22a–b). Rotation is first applied to the body, which is then followed by a translation.

4.1.4. Rigid Body Transformation about a Center Point of the Body

As opposed to a rotation of the body about the origin followed by a translation, let us now look at how to rotate a body about a central point, followed by a translation. To demonstrate this process, we will use Figure 4.4 as a guide.

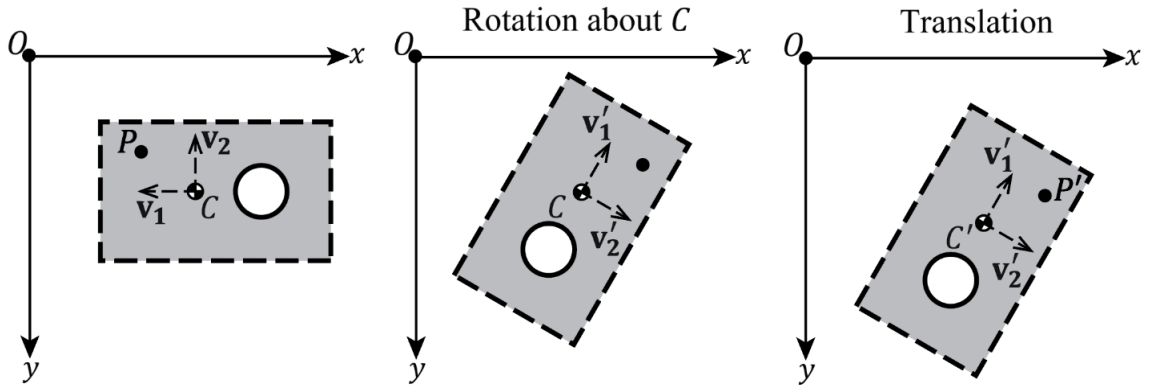


Figure 4.4: Transformation procedure where we first rotate the body about point C, with coordinates $(\mathbf{x_C}, \mathbf{y_C})$, and then translate that body a desired amount. On the left, we start with the body from Figure 4.3 (left) and add two body vectors, $\mathbf{v_1}$ and $\mathbf{v_2}$, that have their tails at C and their heads tangentially pointed toward the rectangle sides. In the middle, we apply a rotation of the body about C which rotates the body vectors to a desired angle, $\mathbf{v_1'}$ and $\mathbf{v_2'}$. On the right, we apply a translation to the body to arrive at our desired, final orientation.

The rotation applied to the body is the same as what is presented in equation (4.4). To apply this rotation to the body about C, we multiply each two-component, body vector, $[\mathbf{v_1}]_2$ and $[\mathbf{v_2}]_2$, with the 2x2 rotation matrix to produce their rotated, body vector counterparts, $[\mathbf{v_1'}]_2$ and $[\mathbf{v_2'}]_2$:

$$[\mathbf{v_1'}]_2 = [\mathbf{v_1}]_2[\mathbf{R}]_2 \tag{4.23a}$$

$$[\mathbf{v_2'}]_2 = [\mathbf{v_2}]_2[\mathbf{R}]_2 \tag{4.23b}$$

Here, $[\mathbf{v_1}]_\mathbf{2}$ is a two-component body vector whose tail is at C and head is tangentially pointed at a rectangle side. The perpendicular body vector, $[\mathbf{v_2}]_\mathbf{2}$, also has its tail at C, but its head pointing at the adjacent side of the rectangle. Therefore, each of the initial body vectors in equations (4.23a-b) are:

$$[\mathbf{v_1}]_\mathbf{2} = [x_{v_1} - x_C \quad y_{v_1} - y_C] \tag{4.24a}$$

$$[\mathbf{v_2}]_\mathbf{2} = [x_{v_2} - x_C \quad y_{v_2} - y_C] \tag{4.24b}$$

The body vectors, post-rotation, are:

$$[\mathbf{v_1'}]_\mathbf{2} = [x'_{v_1} - x_C \quad y'_{v_1} - y_C] \tag{4.25a}$$

$$[\mathbf{v_2'}]_\mathbf{2} = [x'_{v_2} - x_C \quad y'_{v_2} - y_C] \tag{4.25b}$$

By substituting equations (4.24-4.25) into equations (4.23a-b) and following the steps used to derive equation (4.6), we get:

$$\left[x'_{v_1}\ y'_{v_1}\right] = \left[x_{v_1}\ y_{v_1}\right][\mathbf{R}]_\mathbf{2} + [x_C\ y_C] - [x_C\ y_C][\mathbf{R}]_\mathbf{2} \tag{4.26a}$$

$$\left[x'_{v_2}\ y'_{v_2}\right] = \left[x_{v_2}\ y_{v_2}\right][\mathbf{R}]_\mathbf{2} + [x_C\ y_C] - [x_C\ y_C][\mathbf{R}]_\mathbf{2} \tag{4.26b}$$

Equations (4.26a-b) provide the new, body vector locations following a rotation about C.

We now follow-up with a translation of C by adding the vector $[\mathbf{t}]_\mathbf{2} = [t_x\ t_y]$:

$$[\mathrm{x}'_\mathrm{C}\ \mathrm{y}'_\mathrm{C}] = [\mathrm{x}_\mathrm{C}\ \mathrm{y}_\mathrm{C}] + [\mathbf{t}]_\mathbf{2} \tag{4.27}$$

With a new location of C, we redefine equations (4.25) to account for the change to the body vector tails:

$$[\mathbf{v}'_\mathbf{1}]_\mathbf{2} = [x'_{v_1} - x'_C \quad y'_{v_1} - y'_C] \tag{4.28a}$$

$$[\mathbf{v}'_\mathbf{2}]_\mathbf{2} = [x'_{v_2} - x'_C \quad y'_{v_2} - y'_C] \tag{4.28b}$$

We substitute equation (4.27) into equations (4.28a-b):

$$[\mathbf{v}'_\mathbf{1}]_\mathbf{2} = \left[x'_{v_1} - (x_C + t_x) \quad y'_{v_1} - (y_C + t_y)\right] \tag{4.29a}$$

$$[\mathbf{v}'_\mathbf{2}]_\mathbf{2} = \left[x'_{v_2} - (x_C + t_x) \quad y'_{v_2} - (y_C + t_y)\right] \tag{4.29b}$$

By following the steps to derive equations (4.26a-b) but using equations (4.29a-b) instead of (4.25a-b), we get:

$$\left[x'_{v_1}\ y'_{v_1}\right] = \left[x_{v_1}\ y_{v_1}\right][\mathbf{R}]_2 + [x_C\ y_C] - [x_C\ y_C][\mathbf{R}]_2 + [t_x\ t_y] \tag{3.30a}$$

$$\left[x'_{v_2}\ y'_{v_2}\right] = \left[x_{v_2}\ y_{v_2}\right][\mathbf{R}]_2 + [x_C\ y_C] - [x_C\ y_C][\mathbf{R}]_2 + [t_x\ t_y] \tag{3.30b}$$

As is evident when comparing equations (4.26) and (3.30), in the latter, we have only added the translation vector and once again see a two-step process where, in this instance, we rotate the body about C and then translate the rotated body.

4.2. Transformation Matrix for Rigid Registration of Two Bodies

Now, what if we are given two configurations of a rigidly moved body and want to determine the transformation matrix that maps one configuration to another? To determine the transformation matrix that registers, or aligns, the two bodies, we find $[\mathbf{R}]_{\mathbf{4}}$ and $[\mathbf{T}]_{\mathbf{4}}$ by using moments of mass.

We derive the rotation matrix from both the first, second and the third moments of mass. We use the first moment of mass to derive the mass centroid. Eigen decomposition of the second moment, or inertia matrix, about the mass centroid is used to find the principal axes of each body. The third moment about the mass centroid is used to determine a consistent directionality for the principal axes of both bodies. These steps provide rotation matrices that map each body's configuration to one that is aligned with the coordinate system. We will call these rotation matrices "principal inertia matrices". The final rotation matrix, which rotates one body configuration to the other, is derived by bridging these principal inertia matrices from above.

We derive the translation vector by aligning the mass centroid of one configuration to the post-rotated mass centroid (rotation matrix derived from second and third moments of mass) of the other.

4.2.1. Calculation of Rotation Matrix from Moments of Mass

First, we must find the mass centroid:

$$\bar{x} = M_x/M \tag{4.31a}$$

$$\bar{y} = M_y/M \tag{4.31b}$$

$$\bar{z} = M_z/M \tag{4.31c}$$

where M represents the total mass, or the zeroth moment of mass:

$$M = \int_x \int_y \int_z \rho(x,y,z) dz dy dx \tag{4.32}$$

and M_x, M_y, and M_z represent the first moments of mass about each axis:

$$M_x = \int_x \int_y \int_z x\rho(x,y,z) dz dy dx \tag{4.33a}$$

$$M_y = \int_x \int_y \int_z y\rho(x,y,z) dz dy dx \tag{4.33b}$$

$$M_z = \int_x \int_y \int_z z\rho(x,y,z) dz dy dx \tag{4.33c}$$

$\rho(x,y,z)$ represents a 3-D mass density function.

Next, we find the inertia matrix of the body about its mass centroid and perform Eigen decomposition of that matrix. The symmetric, inertia matrix about the mass centroid defines a symmetric 3x3 matrix which describes the inertia distribution of the body, (denoted below as $\mathbf{Inertia_A}$ and $\mathbf{Inertia_B}$) for each configuration, A and B:

$$\mathbf{Inertia_A} = \begin{bmatrix} Inertia_{xx_A} & Inertia_{xy_A} & Inertia_{xz_A} \\ Inertia_{xy_A} & Inertia_{yy_A} & Inertia_{yz_A} \\ Inertia_{xz_A} & Inertia_{yz_A} & Inertia_{zz_A} \end{bmatrix} \tag{4.34a}$$

$$\mathbf{Inertia_B} = \begin{bmatrix} Inertia_{xx_B} & Inertia_{xy_B} & Inertia_{xz_B} \\ Inertia_{xy_B} & Inertia_{yy_B} & Inertia_{yz_B} \\ Inertia_{xz_B} & Inertia_{yz_B} & Inertia_{zz_B} \end{bmatrix} \tag{4.34b}$$

Each of the components in the inertia matrices (4.34a-b) is determined with the following volumetric integrals:

$$Inertia_{xx_{A\,or\,B}} = \int_x \int_y \int_z (x - \bar{x})^2 \rho(x, y, z) dz dy dx \tag{4.35a}$$

$$Inertia_{yy_{A\,or\,B}} = \int_x \int_y \int_z (y - \bar{y})^2 \rho(x, y, z) dz dy dx \tag{4.35b}$$

$$Inertia_{zz_{A\,or\,B}} = \int_x \int_y \int_z (z - \bar{z})^2 \rho(x, y, z) dz dy dx \tag{4.35c}$$

$$Inertia_{xy_{A\,or\,B}} = \int_x \int_y \int_z (x - \bar{x})(y - \bar{y}) \rho(x, y, z) dz dy dx \tag{4.35d}$$

$$Inertia_{xz_{A\,or\,B}} = \int_x \int_y \int_z (x - \bar{x})(z - \bar{z})\rho(x, y, z)dzdydx \tag{4.35e}$$

$$Inertia_{yz_{A\,or\,B}} = \int_x \int_y \int_z (y - \bar{y})(z - \bar{z})\rho(x, y, z)dzdydx \tag{4.35f}$$

In equations (4.35a-f) we integrate over the volumes of the bodies, configuration A or B. Note that because distance is squared in each integral, the sign, or direction, is lost. It is for this reason that we need to use the third moment of mass later. We will cover this further later.

Then, we calculate the eigenvectors that compose the principal inertia matrix through Eigen decomposition of the inertia matrix of the body in the initial configuration (configuration A):

$$[\mathbf{Inertia_A}]\mathbf{u} = \lambda_u \mathbf{u} \tag{4.36a}$$

$$[\mathbf{Inertia_A}]\mathbf{v} = \lambda_v \mathbf{v} \tag{4.36b}$$

$$[\mathbf{Inertia_A}]\mathbf{w} = \lambda_w \mathbf{w} \tag{4.36c}$$

and the following configuration (configuration B):

$$[\mathbf{Inertia_B}]\mathbf{u}' = \lambda_u \mathbf{u}' \tag{4.37a}$$

$$[\mathbf{Inertia_B}]\mathbf{v}' = \lambda_v \mathbf{v}' \tag{4.37b}$$

$$[\mathbf{Inertia_B}]\mathbf{w}' = \lambda_w \mathbf{w}' \tag{4.37c}$$

where $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$ and $\mathbf{u}'$, $\mathbf{v}'$, $\mathbf{w}'$ are the triplet of eigenvectors for configurations A and B, respectively. The eigenvalues λ_u, λ_v, *and* λ_w derived from configuration A are equal to those from B as there is no change in mass distribution, other than direction, of the body between configurations. Each triplet of eigenvectors is a set of orthonormal vectors that align with the principal directions of mass distribution. Together, the triplets compose the columns of 3x3 principal inertia matrices:

$$\mathbf{R_A} = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \tag{4.38a}$$

$$\mathbf{R_B} = \begin{bmatrix} u'_x & v'_x & w'_x \\ u'_y & v'_y & w'_y \\ u'_z & v'_z & w'_z \end{bmatrix} \tag{4.38b}$$

$\mathbf{R_A}$ and $\mathbf{R_B}$ rotate a body from its principal axes to the coordinate axes. Let us demonstrate this behavior with $\mathbf{R_A}$. We do so by rotating each eigenvector that composes the principal inertia matrix by $\mathbf{R_A}$, which is equivalent to multiplying the transpose of $\mathbf{R_A}$ by itself:

$$\mathbf{R_A^T R_A} = \begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{bmatrix} \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \tag{4.39}$$

Carrying out this multiplication yields:

$$\mathbf{R_A^T R_A} = \begin{bmatrix} \left|\left|\mathbf{u}\right|\right|^2 & \mathbf{u} \cdot \mathbf{v} & \mathbf{u} \cdot \mathbf{w} \\ \mathbf{v} \cdot \mathbf{u} & \left|\left|\mathbf{v}\right|\right|^2 & \mathbf{v} \cdot \mathbf{w} \\ \mathbf{w} \cdot \mathbf{u} & \mathbf{w} \cdot \mathbf{v} & \left|\left|\mathbf{z}\right|\right|^2 \end{bmatrix} \tag{4.40}$$

By recalling that the dot product of orthogonal vectors is zero, and that the magnitude of normalized vectors is one, we simplify equation (4.40) to an identity matrix:

$$\mathbf{R_A^T R_A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.41}$$

Equation (4.41) demonstrates that the eigenvectors, after rotating with $\mathbf{R_A}$, point in the direction of the coordinate axes.

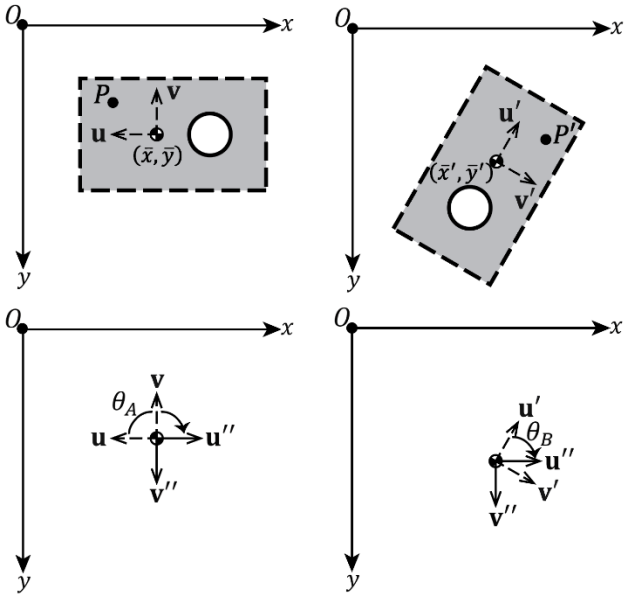


Figure 4.5: 2D representation of inertia axes about the mass centroid on both body orientations (top). The principal inertia axes are designated with u, v and u′, v′ in the left and right configuration, respectively. These perpendicular axes point from the mass centroid, to different rectangle sides. The eigenvector matrix serves as a rotation matrix that rotates the principal inertia axes from their current orientation to one that is parallel to the coordinate axes (bottom).

To illustrate this, let us use Figure 4.5. The principal inertia/rotation matrix for each configurations is:

$$[\mathbf{R_A}]_\mathbf{z} = \begin{bmatrix} cos(\theta_A) & sin\,(\theta_A) & 0 \\ -sin(\theta_A) & cos\,(\theta_A) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.42a}$$

$$[\mathbf{R_B}]_\mathbf{z} = \begin{bmatrix} cos(\theta_B) & sin\,(\theta_B) & 0 \\ -sin(\theta_B) & cos\,(\theta_B) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.42b}$$

Before calculating the final rotation matrix, one more step is needed to ensure that the principal axes are pointing in a consistent direction regardless of body configuration. To further explain, when deriving the inertia matrix, the sign, or directionality of the inertia components was lost due to squaring the distance measures. Regardless, these vectors are parallel to the directions of highest mass distribution. However, they can point in either the positive or negative direction (Figure 4.6).

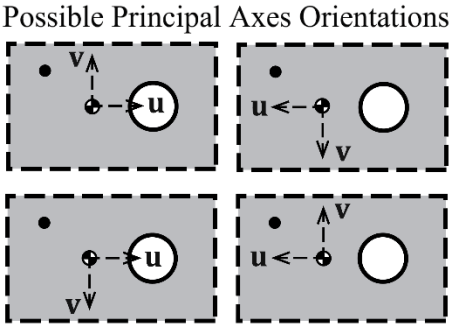


Figure 4.7: Possible orientations for the principal axes calculated from just the Eigen decomposition of the inertia matrix. All possible outcomes are parallel to the distribution of the area in the body. However, that direction can differ in direction, making four possible outcomes in the 2D case depicted, where we impose the restriction that the third eigenvector, $\mathbf{w}$, points into the page. However, if we impose right-handedness on the eigenvector triplet, only the bottom two cases become valid options.

In a 2-D case, this provides four possible vector pairs. We limit our vector pair possibilities to two by imposing right-handedness to the vector combination:

$$(\mathbf{u} \times \mathbf{v}) \cdot \mathbf{w} > 0 \tag{4.43}$$

Note that the operation in equation (4.43) is equivalent to calculating the determinant of a matrix composed of transposes of the eigenvectors:

$$(\mathbf{u} \times \mathbf{v}) \cdot \mathbf{w} = \left\| \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \right\| \tag{4.44}$$

In 3-D, after imposing right-handedness to the eigenvectors, there are four possible vector triplets. With the initial eigenvector orientation, and three 180° rotations about each axis, we derive those four possibilities. Those three rotation matrices are:

$$\mathbf{R}_{\mathbf{180^\circ \, about \, x}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{4.45a}$$

$$\mathbf{R}_{\mathbf{180^\circ \, about \, y}} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{4.45b}$$

$$\mathbf{R}_{\mathbf{180^\circ\ about\ z}} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.45c}$$

To choose the correctly rotated principal inertia matrix, we use the third moment of mass. The third moment of mass, **T**, is a 3x3x3 matrix. **T** can be represented as a 3x3 matrix of vectors:

$$\mathbf{T} = \begin{bmatrix} \mathrm{T_{ixx}} & \mathrm{T_{ixy}} & \mathrm{T_{ixz}} \\ \mathrm{T_{iyx}} & \mathrm{T_{iyy}} & \mathrm{T_{iyz}} \\ \mathrm{T_{izx}} & \mathrm{T_{izy}} & \mathrm{T_{izz}} \end{bmatrix} \tag{4.46}$$

In equation (4.46), each matrix component is a vector whose components are indexed with the variable i. The variable i steps through the integer values of one to three. The matrix components of the third moment of mass are derived with the following volumetric integral:

$$\mathrm{T_{ijk}} = \int_{\mathrm{x}} \int_{\mathrm{y}} \int_{\mathrm{z}} \mathrm{x_i x_j x_k} \rho(\mathrm{x, y, z}) \mathrm{dz dy dx} \tag{4.47}$$

where i, j, and k are indexing variables that count from one to three, and $x_1 = x$, $x_2 = y$, and $x_3 = z$. For example, for $i = 1$, $j = 2$, and $k = 3$:

$$\begin{aligned} T_{123} &= \int_x \int_y \int_z x_1 x_2 x_3 \rho(x, y, z) dz dy dx \\ &= \int_x \int_y \int_z xyz\ \rho(x, y, z) dz dy dx\ = T_{xyz} \end{aligned} \tag{4.48}$$

In our case we want to calculate **T** about the mass centroid. Continuing our example, T_{123}

is calculated about the mass centroid as so:

$$T_{123} = \int_x \int_y \int_z (x - \bar{x})(y - \bar{y})(z - \bar{z})\rho(x, y, z)dzdydx \tag{4.49}$$

With the distance raised to the third power in **T**, we maintain sign, or directionality, of mass distribution. However, to identify the correct eigenvector orientation, we only need to use some of the components of **T**. We use the trace of each plane of **T** in equation (4.46) to define the components of a vector, **k,** that contains information about the third moment of mass corresponding to all three axes. Therefore, we define:

$$\mathbf{k} = [k_x \quad k_y \quad k_z] \tag{4.50}$$

where k_x, k_y, and k_z represent the traces of each plane from equation (4.46) when $\mathrm{i} = 1$, 2, and 3, respectively:

$$\mathrm{k_x} = T_{111} + T_{122} + T_{133} \tag{4.51a}$$

$$\mathrm{k_y} = T_{211} + T_{222} + T_{233} \tag{4.51b}$$

$$\mathrm{k_z} = T_{311} + T_{322} + T_{333} \tag{4.51c}$$

We then test the four possible eigenvector orientations with the following series of equations:

$$d_1 = ||\mathbf{k_A R_A} - \mathbf{k_B R_B}|| \tag{4.52a}$$

$$\mathrm{d_2} = ||\mathbf{k_A R_A} - \mathbf{R_{180^\circ\ about\ x} k_B R_B}|| \tag{4.52b}$$

$$d_3 = ||\mathbf{k_A R_A} - \mathbf{R_{180^\circ\ about\ y} k_B R_B}||$$ (4.52c)

$$d_4 = ||\mathbf{k_A R_A} - \mathbf{R_{180^\circ\ about\ z} k_B R_B}||$$ (4.52d)

where $\mathbf{R_A}$ and $\mathbf{R_B}$ represent the principal inertia matrices and $\mathbf{k_A}$ and $\mathbf{k_B}$ represent the third moment vectors for configurations A and B, respectively. The remaining, "test" rotation matrices are presented in equations (4.45a-d) and $\mathbf{d} = [d_1\ d_2\ d_3\ d_4]$ is a vector whose smallest component identifies the correct "test" rotation matrix, $\mathbf{R_{correct}}$, to apply to $\mathbf{R_B}$ so that it aligns consistently with $\mathbf{R_A}$:

$$\mathbf{R_{Bcorrect}} = \mathbf{R_{correct} R_B}$$ (4.53)

The final rotation matrix, $\mathbf{R_{AB}}$, which rotates configuration A to B, is calculated with:

$$\mathbf{R_{AB}} = \mathbf{R_A R_{Bcorrect}^T}$$ (4.54)

4.2.2. Calculation of Translation Matrix from Moments of Mass

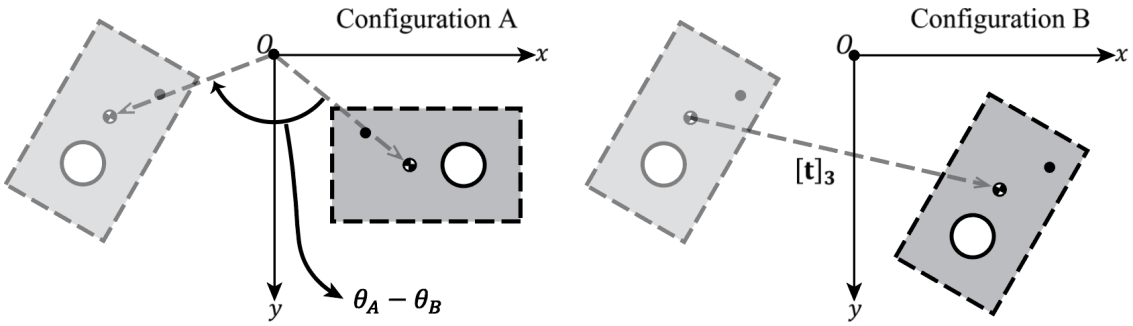


Figure 4.8: A body in its initial configuration (A) is rotated by an angle of $\theta_A - \theta_B$ about the origin (z-axis goes into page) to an intermediate configuration (translucent). The body is then translated by the vector $[\mathbf{t}]_3$ from the intermediate to its final configuration (B).

The translation matrix, which maps the post-rotated body to the desired, final configuration (see Figure 4.7), is calculated by tracking a single point from one configuration to the other.

We use both the initial and new point coordinates, along with the rotation matrix, to calculate the translation vector. We solve for the translation vector components in equations (4.19a–c):

$$t_x = x' - (x * R_{11} + y * R_{21} + z * R_{31}) \tag{4.55a}$$

$$t_y = y' - (x * R_{12} + y * R_{22} + z * R_{32}) \tag{4.55b}$$

$$t_z = z' - (x * R_{13} + y * R_{23} + z * R_{33}) \tag{4.55c}$$

We rewrite equations (4.55a–c) as a single equation:

$$[\mathbf{t}]_3 = [\mathrm{x'y'z'}] - [\mathrm{x\ y\ z}][\mathbf{R}]_3 \tag{4.56}$$

The special point we choose to track throughout rigid motion is the mass centroid as it is easy to calculate and we can feel confident that we are looking at the same point between configurations. In other words, the mass centroid is a material point whose coordinates we can follow regardless of configuration. Therefore, equation (4.56) for the mass centroid is written as:

$$[\mathbf{t}]_{\mathbf{3}} = [\bar{x}'\ \bar{y}'\ \bar{z}'] - [\bar{x}\ \bar{y}\ \bar{z}][\mathbf{R}]_{\mathbf{3}} \tag{4.57}$$

Where $[\bar{x}\ \bar{y}\ \bar{z}]$ and $[\bar{x}'\ \bar{y}'\ \bar{z}']$ represent the vector from the origin to the mass centroid in the initial and rigidly moved locations, respectively. Note that this equation is similar to equation (4.7), where we have replaced the point C with the mass centroid coordinates. We describe how to compute the mass centroid and inertia matrix in the next few sections.

4.3. Determination of Transformation Matrix Using Image Intensity as Mass Density

4.3.1. Calculation of First Moment of Mass (Mass Centroid of Image Intensity)

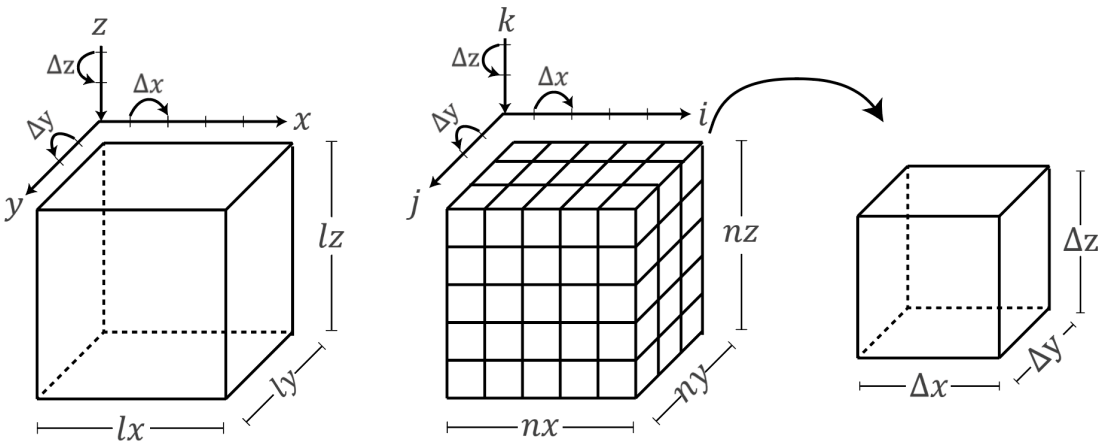


Figure 4.9: Cuboid located in an image coordinate system (left). Here, the origin is located at a slight offset (half-voxel side-length in x, y and z) from the top left corner of the cuboid. The dimensions of the cuboid are large enough to completely enclose the body of interest. This cuboid is then discretized (middle). The discretized cuboid shares the same image coordinate system. Each subvolume (right) is also a cuboid shape, but with side-lengths of Δx=Δy=Δz=1 voxel side-length. The dimensions of the discretized cuboid are nx=lx/Δx, ny=ly/Δy, and nz=lz/Δz.

We start by calculating the mass of a cuboid that will later represent a 3D image (Figure 4.8). To facilitate this forthcoming transition, we place this cuboid in an image coordinate system where tick spacing is equal to a voxel side-length: Δx, Δy, and Δz. The origin of said coordinate system is offset as similarly seen in Figure 3.6 (a half-voxel side length offset from the top left corner of the cuboid). This cuboid will represent the field of view (FOV) that captures the body of interest. Therefore, the cuboid's side lengths, lx, ly, and lz, are the dimensions of the image's FOV. Finally, we note that if there is no matter at a point in the cuboid/FOV, the density of that region is zero.

Having established our coordinate system and dimensions of our cuboid, we calculate its mass with an integration of its mass density over the whole volume:

$$M = \int_{z=0.5\Delta z}^{lz+0.5\Delta z} \int_{y=0.5\Delta y}^{ly+0.5\Delta y} \int_{x=0.5\Delta x}^{lx+0.5\Delta x} \rho(x,y,z)dxdydz \tag{4.58}$$

where M is the total mass of the object, ρ is the mass density, and we integrate over the volume of the object. The limits in equation (4.58) follow the convention presented in Figure 4.8 and its description. We approximate that volume integral as a Riemann sum over small volumes:

$$M \approx \sum_{k=1}^{nz} \sum_{j=1}^{ny} \sum_{i=1}^{nx} \rho\left(x_i, y_j, z_k\right)\Delta x\Delta y\Delta z \tag{4.59}$$

Here, Δx, Δy, and Δz are the voxel side-lengths and nx, ny, and nz are the number of voxels in the x, y, and z directions, respectively. The number of voxels are calculated as so:

$$nx = lx/\Delta x \tag{4.60a}$$

$$ny = ly/\Delta y \tag{4.60b}$$

$$nz = lz/\Delta z \tag{4.60c}$$

By discretizing (4.58) as a Riemann sum, we now sample ρ with x_i, y_j, and z_k. Let

$$x_i = i\Delta x \tag{4.61a}$$

$$y_j = j\Delta y \tag{4.61b}$$

$$z_k = k\Delta z \tag{4.61c}$$

where i, j, and k are indexed positions along the x-, y-, and z-axis, respectively. For example, we specify that $i = 1, 2, 3, \ldots, nx$ corresponds to the coordinates: $x_1 = \Delta x$, $x_2 = 2\Delta x$, $x_3 = 3\Delta x$, ..., $x_{nx} = nx\Delta x$. To further clarify, the coordinate x_1 would be the left-most part of the cuboid. The same convention is followed for y_j and z_k equations (4.61b) and (4.61c). In this way, the sampled values of ρ become representative of their respective subvolumes. Therefore, ρ becomes a 3D matrix, ρ_{ijk}, with a single measure of density at each matrix component:

$$\rho(x_i, y_j, z_k) = \rho_{ijk} \tag{4.62}$$

By substituting equation (4.62), (4.59) becomes:

$$M \approx \sum_{k=1}^{nz} \sum_{j=1}^{ny} \sum_{i=1}^{nx} \rho_{ijk} \Delta x \Delta y \Delta z \tag{4.63}$$

For finding the total "mass" of an image, we consider that image voxel intensity is analogous to mass density. In μCT scans, a voxel intensity value is driven, approximately, by the density of the material contained in that voxel during the scan. To further make the analogy seamless, we make the assumption that all voxel-intensities are non-negative. This is a valid assumption as intensity values are a measure of x-ray absorption and negative values in the physical world would indicate that a region of the object scanned is emitting x-rays, which is unlikely [17]. Instead, negative values are typically artifacts introduced during image reconstruction. As negative values are an unwanted reconstruction artifact,

our assumption is sustained by truncating all negative values.

Using this analogy, total "mass" of an image, M_I, is found by summing intensity values over the entire image matrix, I_{ijk}, where each component in I_{ijk} is a voxel and contains a grayscale intensity value. In other words, I_{ijk} is intensity per unit volume, where the unit is voxel in 3D or pixel in 2D. By replacing the density variable, ρ_{ijk}, with image intensity, I_{ijk}, we get:

$$M_I = \sum_{k=1}^{nz} \sum_{j=1}^{ny} \sum_{i=1}^{nx} I_{ijk} \Delta x \Delta y \Delta z \tag{4.64}$$

For an image, nx, ny, and nz represent the image dimensions, or number of voxels, in each axis. The volume of an element is one cubic voxel:

$$\Delta x \Delta y \Delta z = 1\ (voxel\ side\ length)^3 \tag{4.65}$$

Therefore,

$$M_I = \sum_{k=1}^{nz} \sum_{j=1}^{ny} \sum_{i=1}^{nx} I_{ijk} \tag{4.66}$$

Keep in mind that although we have simplified M_I, its units will end up being some measure of mass. The exact units will depend on how the voxel values are represented. For example, bone density is typically in units of Hydroxyapatite density ($mg\ HA/cm^3$) and if we convert voxel volume to units of cm^3, we would be left with the total bone mass expressed in units of $mg\ HA$.

Now, to calculate the center of mass of the cuboid, we first calculate the first moment of mass, which is defined for each coordinate axis as:

$$M_x = \int_{x=0.5\Delta x}^{lx+0.5\Delta x} \int_{y=0.5\Delta y}^{ly+0.5\Delta y} \int_{z=0.5\Delta z}^{lz+0.5\Delta z} (x - x_0)\rho(x,y,z)dzdydx \qquad (4.67a)$$

$$M_y = \int_{x=0.5\Delta x}^{lx+0.5\Delta x} \int_{y=0.5\Delta y}^{ly+0.5\Delta y} \int_{z=0.5\Delta z}^{lz+0.5\Delta z} (y - y_0)\rho(x,y,z)dzdydz \qquad (4.67b)$$

$$M_z = \int_{x=0.5\Delta x}^{lx+0.5\Delta x} \int_{y=0.5\Delta y}^{ly+0.5\Delta y} \int_{z=0.5\Delta z}^{lz+0.5\Delta z} (z - z_0)\rho(x,y,z)dzdydz \qquad (4.67c)$$

where M_x, M_y, and M_z represent the moments along each coordinate axis about the coordinates *(*x_0, y_0, and z_0). If we want to find the mass centroid coordinates *(*$\bar{x}$, $\bar{y}$, and $\bar{z}$*)* with respect to the origin of our coordinate system, we set x_0, y_0, and z_0 equal to zero and equations (4.67a-c) become:

$$M_x = \int_{\mathrm{x}=0.5\Delta \mathrm{x}}^{\mathrm{lx}+0.5\Delta \mathrm{x}} \int_{y=0.5\Delta y}^{ly+0.5\Delta y} \int_{z=0.5\Delta z}^{lz+0.5\Delta z} x\rho(x,y,z)dzdydx \qquad (4.68a)$$

$$M_y = \int_{x=0.5\Delta x}^{lx+0.5\Delta x} \int_{y=0.5\Delta y}^{ly+0.5\Delta y} \int_{z=0.5\Delta z}^{lz+0.5\Delta z} y\rho(x,y,z)dzdydx \qquad (4.68b)$$

$$M_z = \int_{x=0.5\Delta x}^{lx+0.5\Delta x} \int_{y=0.5\Delta y}^{ly+0.5\Delta y} \int_{z=0.5\Delta z}^{lz+0.5\Delta z} z\rho(x,y,z)dzdydx \qquad (4.68c)$$

Keeping basic mechanical engineering principles in mind, equations (4.68a–c) can be seen as three dimensions of mechanical moments of a force from the origin. Except in this case, the mass of the object is its force. For example, in equation (4.68a), the first integral ($\int_{x=0.5\Delta x}^{lx+0.5\Delta x} x dx$) is the x-distance from point of rotation (in this case the origin), and the second half ($\int_{y=0.5\Delta y}^{ly+0.5\Delta y} \int_{z=0.5\Delta z}^{lz+0.5\Delta z} \rho(x,y,z) dz dy$) is the mass of a slice of the cuboid at the specified x-coordinate.

We take the fraction of each moment over the total mass to calculate the mass centroid coordinates with respect to the origin as seen in equations (4.31a–c). Then, similar to when we approximated (4.58) with (4.59), we use a Riemann sum to approximate (4.68a–c) as:

$$M_x \approx \sum_{i=1}^{nx} \sum_{j=1}^{ny} \sum_{k=1}^{nz} x_i \rho(x_i, y_j, z_k) \Delta z \Delta y \Delta x \tag{4.69a}$$

$$M_y \approx \sum_{i=1}^{nx} \sum_{j=1}^{ny} \sum_{k=1}^{nz} y_j \rho(x_i, y_j, z_k) \Delta z \Delta y \Delta x \tag{4.69b}$$

$$M_z \approx \sum_{i=1}^{nx} \sum_{j=1}^{ny} \sum_{k=1}^{nz} z_k \rho(x_i, y_j, z_k) \Delta z \Delta y \Delta x \tag{4.69c}$$

If we recall equations (4.61a–c), we can rewrite equations (4.69a–c) as:

$$M_x \approx \sum_{i=1}^{nx} \sum_{j=1}^{ny} \sum_{k=1}^{nz} i \Delta x \rho(x_i, y_j, z_k) \Delta z \Delta y \Delta x \tag{4.70a}$$

$$M_y \approx \sum_{i=1}^{nx}\sum_{j=1}^{ny}\sum_{k=1}^{nz} j\Delta y\rho\left(x_i, y_j, z_k\right)\Delta z\Delta y\Delta x \tag{4.70b}$$

$$M_z \approx \sum_{i=1}^{nx}\sum_{j=1}^{ny}\sum_{k=1}^{nz} k\Delta z\rho\left(x_i, y_j, z_k\right)\Delta z\Delta y\Delta x \tag{4.70c}$$

We simplify further by substituting from equation (4.62),

$$M_x \approx \sum_{i=1}^{nx}\sum_{j=1}^{ny}\sum_{k=1}^{nz} i\Delta x\rho_{ijk}\Delta z\Delta y\Delta x \tag{4.71a}$$

$$M_y \approx \sum_{i=1}^{nx}\sum_{j=1}^{ny}\sum_{k=1}^{nz} j\Delta y\rho_{ijk}\Delta z\Delta y\Delta x \tag{4.71b}$$

$$M_z \approx \sum_{i=1}^{nx}\sum_{j=1}^{ny}\sum_{k=1}^{nz} k\Delta z\rho_{ijk}\Delta z\Delta y\Delta x \tag{4.71c}$$

If we rewrite equations (4.71a–c) for image space, similar to deriving (4.66), we get:

$$M_{x_I} \approx \sum_{i=1}^{nx}\sum_{j=1}^{ny}\sum_{k=1}^{nz} i\Delta x I_{ijk}\Delta z\Delta y\Delta x \tag{4.72a}$$

$$M_{y_I} \approx \sum_{i=1}^{nx}\sum_{j=1}^{ny}\sum_{k=1}^{nz} j\Delta y I_{ijk}\Delta z\Delta y\Delta x \tag{4.72b}$$

$$M_{z_I} \approx \sum_{i=1}^{nx}\sum_{j=1}^{ny}\sum_{k=1}^{nz} k\Delta z I_{ijk}\Delta z\Delta y\Delta x \tag{4.72c}$$

Furthermore, Δx, Δy, and Δz equal one voxel side-length, which shortens equations (4.72a–c) to:

$$M_{x_I} \approx \sum_{i=1}^{nx}\sum_{j=1}^{ny}\sum_{k=1}^{nz} iI_{ijk} \tag{4.73a}$$

$$M_{y_I} \approx \sum_{i=1}^{nx}\sum_{j=1}^{ny}\sum_{k=1}^{nz} jI_{ijk} \tag{4.73b}$$

$$M_{z_I} \approx \sum_{i=1}^{nx}\sum_{j=1}^{ny}\sum_{k=1}^{nz} kI_{ijk} \tag{4.73c}$$

We also rewrite (4.31a–c) for image space as:

$$\bar{x}_I = M_{x_I}/M_I \tag{4.74a}$$

$$\bar{y}_I = M_{y_I}/M_I \tag{4.74b}$$

$$\bar{z}_I = M_{z_I}/M_I \tag{4.74c}$$

We now substitute (4.73a–c) into (4.74a–c) and use the total image "mass", M_I, from equation (4.66) to obtain the mass centroid coordinates of an image:

$$\bar{x}_I = \frac{1}{M_I} \sum_{i=1}^{nx} \sum_{j=1}^{ny} \sum_{k=1}^{nz} iI_{ijk} \tag{4.75a}$$

$$\bar{y}_I = \frac{1}{M_I} \sum_{j=1}^{ny} \sum_{i=1}^{nx} \sum_{k=1}^{nz} jI_{ijk} \tag{4.75b}$$

$$\bar{z}_I = \frac{1}{M_I} \sum_{k=1}^{nz} \sum_{i=1}^{nx} \sum_{j=1}^{ny} kI_{ijk} \tag{4.75c}$$

4.3.2. Calculation of Second Moment of Mass (Inertia Matrix of Image Intensity) about Centroid

Now that we have identified the mass centroid, we calculate each component of the inertia matrix, introduced in equation (4.34a), about that mass centroid for the cuboid:

$$Inertia_{xx} = \int_{x=0.5\Delta x}^{lx+0.5\Delta x} \int_{y=0.5\Delta y}^{ly+0.5\Delta y} \int_{z=0.5\Delta z}^{lz+0.5\Delta z} (x-\bar{x})^2 \rho(x,y,z) dz dy dx \tag{4.76a}$$

$$Inertia_{yy} = \int_{x=0.5\Delta x}^{lx+0.5\Delta x} \int_{y=0.5\Delta y}^{ly+0.5\Delta y} \int_{z=0.5\Delta z}^{lz+0.5\Delta z} (y-\bar{y})^2 \rho(x,y,z) dz dx dy \tag{4.76b}$$

$$Inertia_{zz} = \int_{x=0.5\Delta x}^{lx+0.5\Delta x} \int_{y=0.5\Delta y}^{ly+0.5\Delta y} \int_{z=0.5\Delta z}^{lz+0.5\Delta z} (z-\bar{z})^2 \rho(x,y,z) dy dx dz \tag{4.76c}$$

$$Inertia_{xy} = \int_{x=0.5\Delta x}^{lx+0.5\Delta x} \int_{y=0.5\Delta y}^{ly+0.5\Delta y} \int_{z=0.5\Delta z}^{lz+0.5\Delta z} (x-\bar{x})(y-\bar{y})\rho(x,y,z)dzdydx \tag{4.76d}$$

$$Inertia_{xz} = \int_{x=0.5\Delta x}^{lx+0.5\Delta x} \int_{y=0.5\Delta y}^{ly+0.5\Delta y} \int_{z=0.5\Delta z}^{lz+0.5\Delta z} (x-\bar{x})(z-\bar{z})\rho(x,y,z)dydzdx \tag{4.76e}$$

$$Inertia_{yz} = \int_{x=0.5\Delta x}^{lx+0.5\Delta x} \int_{y=0.5\Delta y}^{ly+0.5\Delta y} \int_{z=0.5\Delta z}^{lz+0.5\Delta z} (y-\bar{y})(z-\bar{z})\rho(x,y,z)dxdzdy \tag{4.76f}$$

Using the same steps used to derive (4.66) or (4.75a-c), we both discretize and translate (4.76a-f) into image space to calculate the image inertia matrix components defined about the mass centroid:

$$Inertia_{xx_I} = \sum_{i=1}^{nx}\sum_{j=1}^{ny}\sum_{k=1}^{nz}(i-\bar{x})^2 I_{ijk} \tag{4.77a}$$

$$Inertia_{yy_I} = \sum_{i=1}^{nx}\sum_{j=1}^{ny}\sum_{k=1}^{nz}(j-\bar{y})^2 I_{ijk} \tag{4.77b}$$

$$Inertia_{zz_I} = \sum_{i=1}^{nx}\sum_{j=1}^{ny}\sum_{k=1}^{nz}(k-\bar{z})^2 I_{ijk} \tag{4.77c}$$

$$Inertia_{xy_I} = \sum_{i=1}^{nx}\sum_{j=1}^{ny}\sum_{k=1}^{nz}(i-\bar{x})(j-\bar{y}) I_{ijk} \tag{4.77d}$$

$$Inertia_{xz_I} = \sum_{i=1}^{nx}\sum_{j=1}^{ny}\sum_{k=1}^{nz}(i-\bar{x})(k-\bar{z})I_{ijk} \tag{4.77e}$$

$$Inertia_{yz_I} = \sum_{i=1}^{nx}\sum_{j=1}^{ny}\sum_{k=1}^{nz}(j-\bar{y})(k-\bar{z})I_{ijk} \tag{4.77f}$$

Equations (4.77a-f) are compiled in an image inertia matrix about the mass centroid, $\mathbf{Inertia_I}$:

$$\mathbf{Inertia_I} = \begin{bmatrix} Inertia_{xx_I} & Inertia_{xy_I} & Inertia_{xz_I} \\ Inertia_{xy_I} & Inertia_{yy_I} & Inertia_{yz_I} \\ Inertia_{xz_I} & Inertia_{yz_I} & Inertia_{zz_I} \end{bmatrix} \tag{4.78}$$

4.3.3. Calculation of Third Moment of Mass (Third Moment Vector of Image Intensity) about Centroid

Each component of **k**, the 3x1 third moment vector presented in equations (4.50-4.51), for the cuboid is defined as:

$$k_x = \int_{x=0.5\Delta x}^{lx+0.5\Delta x} \int_{y=0.5\Delta y}^{ly+0.5\Delta y} \int_{z=0.5\Delta z}^{lz+0.5\Delta z} (x-\bar{x})\, r^2\, \rho(x,y,z)\, dzdydx \tag{4.79a}$$

$$k_y = \int_{0.5\Delta x}^{lx+0.5\Delta x} \int_{y=0.5\Delta y}^{ly+0.5\Delta y} \int_{z=0.5\Delta z}^{lz+0.5\Delta z} (y-\bar{y})\, r^2\, \rho(x,y,z)\, dzdxdy \tag{4.79b}$$

$$k_z = \int_{0.5\Delta x}^{lx+0.5\Delta x} \int_{y=0.5\Delta y}^{ly+0.5\Delta y} \int_{z=0.5\Delta z}^{lz+0.5\Delta z} (z-\bar{z})\, r^2\, \rho(x,y,z)\, dydxdz \tag{4.79c}$$

where,

$$r^2 = (x-\bar{x})^2 + (y-\bar{y})^2 + (z-z)^2 \tag{4.80}$$

We now translate equations (4.79-4.80) to discrete, image space:

$$k_{x_I} = \sum_{i=1}^{nx} \sum_{j=1}^{ny} \sum_{k=1}^{nz} (i-\bar{x})\, r_{ijk}^2 I_{ijk} \tag{4.81a}$$

$$k_{y_I} = \sum_{i=1}^{nx} \sum_{j=1}^{ny} \sum_{k=1}^{nz} (j-\bar{y})\, r_{ijk}^2 I_{ijk} \tag{4.81b}$$

$$k_{z_I} = \sum_{i=1}^{nx} \sum_{j=1}^{ny} \sum_{k=1}^{nz} (k - \bar{z})\, r_{ijk}^2 I_{ijk} \tag{4.81c}$$

where,

$$r_{ijk}^2 = (i - \bar{x}_I)^2 + (j - \bar{y}_I)^2 + (k - \bar{z}_I)^2 \tag{4.82}$$

The 3x1 third moment vector (of image intensity) about the mass centroid is:

$$\mathbf{k}_{\mathrm{I}} = [k_{x_I} \quad k_{y_I} \quad k_{z_I}] \tag{4.83}$$

4.3.4. Implementation

We created two MATLAB functions to rigidly register images using moments of mass. The first function, `match_these.m`, receives two image matrices (`I1` and `I2`) and an option call (`rc`), which adapts the function output according to the indexing convention of the input images. The output for `match_these.m` is a rigid transformation matrix, `ttmatAB`, that maps `I1` to `I2`. In our DVC workflow, we use the downsampled (by a factor of five) version of each image (`scanfile_DS` from `downsampleIM.m` explained in section 3.5.3) as `I1` and `I2`, the input images for `match_these.m`.

The second function, `get_inertia.m`, which is used in `match_these.m`, calculates the mass centroid coordinates (`xbar_I`), inertia matrix (`inertia_I`), and third moment vector (`k_I`) of the input image (`im`).

```
%-------------------------------------------------------------------------%
%                                                                         %
% MATLAB function: match_these.m                                          %
%                                                          version 1.2    %
%-------------------------------------------------------------------------%
% Author        : Paul Barbone
% Version       : 1.2
% LastEdit date : 2 June 2020
% Description   : script to rigidly register 3D images
%                 by matching inertia tensors.
%
% History:
% 7 Feb 2019        : Paul Barbone - Creation
% 28 March 2019     : Paul Barbone - Change to function.
% 7  April 2019     : Paul Barbone - Fix for sign ambiguity in
%                                    principal direction transformation.
% 6 August 2019     : Fredy Loaiza - Adapt to only output TMAT
% 2 June 2020       : Fredy Loaiza - Rename variables and comment code
%                                    to include in DVC tutorial
%-------------------------------------------------------------------------%
%                                                                         %
% Boston University                                                       %
%                                                                         %
%-------------------------------------------------------------------------%
%
%-------------------------------------------------------------------------%
% Components/Includes                                                     %
%-------------------------------------------------------------------------%
%
% The functions this code calls:
```

```
%       getinertia.m    Computes image centroid, inertia tensor and
%                       third moment vector
%
%-------------------------------------------------------------------------%
% Input                                                                   %
%-------------------------------------------------------------------------%
%   I1 = 3D matrix of image intensities of one configuration.
%         Ideally thresholded with zeros in background.
%
%   I2 = 3D matrix of image intensities of another configuration.
%         Ideally thresholded with zeros in background.
%
%   rc  = Boolean = 1 if images are stored as I1(x=row,y=col,z=depth).
%                 = 0 if images are stored as I1(y=row,x=col,z=depth),
%                   as Matlab treats image transformations.
%
%-------------------------------------------------------------------------%
% Output                                                                  %
%-------------------------------------------------------------------------%
%   ttmatAB = 4x4 rigid transformation matrix that estimates mapping of
%             I1 to I2.
%-------------------------------------------------------------------------%
function ttmatAB = match_these(I1,I2,rc)

       % Get reference transformation parameters via moments of mass from I1:
           [xbar_I_A, inertia_I_A, k_I_A] = getinertia(I1);
       % Derive eigenvectors of the inertia matrix from I1:
       % Eigenvector matrix represents a rotation from the principal inertia
       % axes of the coordinate axes of the image.
           [R_I_A,~] = eig(inertia_I_A);
```

```
% Make the eigenvector matrix right-handed:
    R_I_A = det(R_I_A)*R_I_A;

% ***NB: Can clear I1 at this point if we need the memory. ***
% clear I1

% Get reference transformation parameters via moments of mass from I2:
    [xbar_I_B, inertia_I_B, k_I_B] = getinertia(I2);
% Derive eigenvectors of the inertia matrix from I2:
    [R_I_B,~] = eig(inertia_I_B);
% Make the eigenvector matrix right-handed:
    R_I_B = det(R_I_B)*R_I_B;

% ***NB: Can clear I2 at this point if we need the memory. ***
% clear I2

% Create pi (180 degree) rotation matrices about each coordinate axis:
    P = zeros(3,3,4);
    P(:,:,1) = eye(3);
    for j = 2:4
      P(:,:,j) = -eye(3);
    end
    P(1,1,2) = 1;   % rotation about the x1 (x) axis.
    P(2,2,3) = 1;   % rotation about the x2 (y) axis.
    P(3,3,4) = 1;   % rotation about the x3 (z) axis.

% Identify the orientation (pi rotation) between the two sets of
% eigenvector matrices:
    d_I = zeros(1,4);
    for j = 1:4
```

```
      d(j) = norm(k_I_A*R_I_A - squeeze(P(:,:,j))* k_I_B*R_I_B);
    end
    sprintf('d for each pi rotation is: %.3e %.3e %.3e %.3e',d)
    % Apply correct pi rotation matrix to R_B
        R_correct = P(:,:,(d_I == min(d_I)));
        R_I_Bcorrect = R_I_B*R_correct;

% COMPUTE TRANSFORMATIONS ----------------------------------------

% Create matrix to switch row and columns for matlab.
% I'm using im(x=row, y=col, z=depth),
% matlab's transformation uses im(y=row, x=col, z=depth).
switch_row_column = eye(4);
switch_row_column(1,1) = 0;
switch_row_column(1,2) = 1;
switch_row_column(2,1) = 1;
switch_row_column(2,2) = 0;

% COMPUTE TRANSFORMATION FROM PRE-DEF TO POST-DEF IMAGES:
ttmatAB = zeros(4,4);
ttmatAB(1:3,1:3) = R_I_A*R_I_Bcorrect';
ttmatAB(:,4) = 0.0;
ttmatAB(4,4) = 1.0;
ttmatAB(4,1:3) = xbar_I_B - xbar_I_A*R_I_A*R_I_Bcorrect';
% ttmat21 = (switch_row_column*ttmat21*switch_row_column);
% tf21 = affine3d(ttmat21);
% tfAB = affine3d(switch_row_column*ttmatAB*switch_row_column);

% COMPUTE TRANSFORMATION FROM POST-DEF TO PRE-DEF IMAGES:
ttmatBA = zeros(4,4);
```

```
        ttmatBA(1:3,1:3) = R_I_Bcorrect*R_I_A';
        ttmatBA(:,4) = 0.0;
        ttmatBA(4,4) = 1.0;
        ttmatBA(4,1:3) = xbar_I_A - xbar_I_B*R_I_Bcorrect*R_I_A';
        % ttmat12 = switch_row_column*ttmat12*switch_row_column;
        % tf12 = affine3d(ttmat12);
        % tfAB = affine3d(switch_row_column*ttmatBA*switch_row_column);

        % Switch transformation matrix convention to convention of row/col
        if (rc==0)
          ttmatBA = switch_row_column*ttmatBA*switch_row_column;
        end
return;
```

```
%-------------------------------------------------------------------------%
%                                                                         %
% MATLAB function: getinertia.m                                           %
%                                                          version 1.1    %
%-------------------------------------------------------------------------%
% Author        : Paul Barbone
% Version       : 1.1
% LastEdit date : 7 February 2019
% Description   : compute centroid and inertia tensor about the centroid.
%
% 7 Feb 2019    : Paul Barbone: Creation
% 28 March 2019 : PB: Modified to use row-column format rather
%           than x-y format.
%
%-------------------------------------------------------------------------%
%                                                                         %
%             PEB: Boston University                                      %
%               Boston, MA                                                %
%                                                                         %
%-------------------------------------------------------------------------%
%
%-------------------------------------------------------------------------%
%               Components/Includes                                       %
%-------------------------------------------------------------------------%
%
% none
%
%-------------------------------------------------------------------------%
%  Input(s)    : image
%  Option(s)   :
```

```
%  Output(s)   : xbar_bold_I = 3x1 centroid of the image im(x1,x2,x3):
%       xbar_bold_I (1) = centroid in x1 direction (row)
%       xbar_bold_I (2) = centroid in x2 direction (column)
%       xbar_bold_I (3) = centroid in x3 direction (depth)
%       inertia_I = 3x3 moment of inertia tensor about xbar_bold _I.
%       k_I (1) = 3rd moment in x1 direction (row)
%       k_I (2) = 3rd moment in x2 direction (column)
%       k_I (3) = 3rd moment in x3 direction (depth)
%  ExitStatus  : none
%-------------------------------------------------------------------------%

function [xbar_bold_I,inertia_I,k_I] = getinertia(im);
      % Extract image size
          [nx,ny,nz] = size(I1);

      % Compute centroid of image intensity:
          % Calculate total intensity of image (zeroth moment of mass)
              d = sum(sum(sum(abs(im),3),2));

          % Calculate the centroid using first moment of mass along each axis
              x = [1:nx]';
              xbar_I = sum(x.*sum(sum(abs(im),3),2))/d;
              y = [1:ny];
              ybar_I = sum(y.*squeeze(sum(sum(abs(im),3),1)))/d;
              z = [1:nz]';
              zbar_I = sum(z.*squeeze(sum(sum(abs(im),2),1)))/d;

              xbar_bold_I = [xbar_I, ybar_I, zbar_I];

      % Compute moments of inertia of image intensity (second moment of mass):
```

```
    inertia_xx                =                sum((kx_I-xbar_I).*(kx_I-
xbar_I).*sum(sum(abs(im),3),2));
    inertia_yy                    =                    sum((y-ybar_I).*(y-
ybar_I).*squeeze(sum(sum(abs(im),3),1)));
    inertia_xy = sum(sum(((kx_I-xbar_I)*(y-ybar_I)).*sum(abs(im),3)));
    inertia_xz                =                sum(sum(((kx_I-xbar_I)*(z-
zbar_I)').*squeeze(sum(abs(im),2))));
    inertia_yz                 =                 sum(sum(((y-ybar_I)'*(z-
zbar_I)').*squeeze(sum(abs(im),1))));
    inertia_zz                    =                    sum((z-zbar_I).*(z-
zbar_I).*squeeze(sum(sum(abs(im),2),1)));
    % Compile inertia matrix components into single 3x3 matrix
        inertia_I (1,1) = inertia_xx;
        inertia_I (1,2) = inertia_xy;
        inertia_I (1,3) = inertia_xz;
        inertia_I (2,1) = inertia_xy;
        inertia_I (2,2) = inertia_yy;
        inertia_I (2,3) = inertia_yz;
        inertia_I (3,1) = inertia_xz;
        inertia_I (3,2) = inertia_yz;
        inertia_I (3,3) = inertia_zz;

% Compute 3rd moment vector of image intensity:
% The apparent difference in the code below from what is presented in the
% tutorial is simply a rearrangement to make the calculations more
% computationally efficient (avoiding FOR loops).
    xx = zeros(nx,1,1);
    yy = zeros(1,ny,1);
    zz = zeros(1,1,nz);
    xx(:,1,1) = [1:nx]-xbar_I;
```

```
            yy(1,:,1) = [1:ny]-ybar_I;
            zz(1,1,:) = [1:nz]-zbar_I;
            % The following is much more computationally efficient:
            im_x_rbar2 = double(abs(im)).*(xx.*xx + yy.*yy + zz.*zz);
            % rbar2 = (xx.*xx + yy.*yy + zz.*zz);
            kx_I = sum((x-xbar_I).*sum(sum(im_x_rbar2,3),2));
            ky_I = sum((y-ybar_I).*sum(sum(im_x_rbar2,3),1));
            kz_I = sum((z-zbar_I).*squeeze(sum(sum(im_x_rbar2,2),1)));

            k_I = zeros(1,3);
            k_I (1) = kx_I;
            k_I (2) = ky_I;
            k_I (3) = kz_I;
return;
```

5. DVC ALGORITHM IN 3D

5.1. Displacement Derivation in 3D

To begin, we present our cost function:

$$C\big(\boldsymbol{u}(\boldsymbol{x})\big) = \frac{1}{2}\int_{\Omega}\big[I_1(\boldsymbol{x}) - I_2\big(\boldsymbol{x}+\boldsymbol{u}(\boldsymbol{x})\big)\big]^2 d\Omega + \frac{\alpha}{2}\int_{\Omega}\boldsymbol{\epsilon}\colon\boldsymbol{\epsilon}\, d\Omega \tag{5.1}$$

This cost function is similar to what we see in equation (1.40) but with two key differences: the integration domain and regularization in the second term. The integration domain is now along three axes ($x, y,$ and z represented as a single vector $\boldsymbol{x}$). For efficiency, instead of writing the triple integral $\int_x\int_y\int_z dzdydx$, we use $\int_\Omega d\Omega$. As for regularization, whereas in equation (1.40) we regularize the displacement gradient, in equation (5.1) we regularize infinitesimal strain, $\boldsymbol{\epsilon}$, *i.e.* we limit large strains. By penalizing strains rather than the displacement gradient, we avoid penalizing small, rigid rotations. In our quasi-static, vertebral fracture experiments, we maintain small, non-rigid deformations by applying small displacement-loading increments to the samples.

Following the derivation presented in sections 1.2.1 and 1 is non-trivial in 3D. Therefore, we will go through it now. We will begin by recalling the strain matrix in terms of displacement.

$$\boldsymbol{\epsilon} = \frac{1}{2}\big(\boldsymbol{\nabla}\mathbf{u} + \boldsymbol{\nabla}\mathbf{u}^{\mathrm{T}}\big) \tag{5.2}$$

Where

$$\nabla \mathbf{u} = \begin{bmatrix} \frac{\partial u_1}{\partial x_1} & \frac{\partial u_2}{\partial x_1} & \frac{\partial u_3}{\partial x_1} \\ \frac{\partial u_1}{\partial x_2} & \frac{\partial u_2}{\partial x_2} & \frac{\partial u_3}{\partial x_2} \\ \frac{\partial u_1}{\partial x_3} & \frac{\partial u_2}{\partial x_3} & \frac{\partial u_3}{\partial x_3} \end{bmatrix} \tag{5.3}$$

and $\nabla \mathbf{u}^T$ is the transpose of (5.3). Here we note that we use indices 1,2,3, corresponding to the x,y,z directions, respectively. In particular x1 = x, x2 = y, and x3 = z.

Note that in index notation, we rewrite (5.2) as:

$$\epsilon_{ij} = \frac{1}{2}(\partial_i u_j + \partial_j u_i) \tag{5.4}$$

With this notation, we step through each indexing variable three times (i.e. i and j each step through the values $1, 2,$ and 3).

Moving forward, let us recall that we begin our $\mathbf{u}(\mathbf{x})$ derivation by defining:

$$\mathbf{u}(\mathbf{x}) = \mathbf{u}^*(\mathbf{x}) + \beta \mathbf{w}(\mathbf{x}) \tag{5.5}$$

Where we recall that $\mathbf{u}^*(\mathbf{x})$ is the (unknown) minimizer of C that we seek to find. We substitute equation (5.5) into (5.1):

$$\begin{aligned} C(\mathbf{u}^* + \beta \mathbf{w}) = \frac{1}{2}\int_\Omega [I_1(\boldsymbol{x}) - I_2(\boldsymbol{x} + \mathbf{u}^* + \beta \mathbf{w})]^2 d\Omega \\ + \frac{\alpha}{2}\int_\Omega [\boldsymbol{\epsilon}(\boldsymbol{u}^*) + \beta \boldsymbol{\epsilon}(\boldsymbol{w})] : [\boldsymbol{\epsilon}(\boldsymbol{u}^*) + \beta \boldsymbol{\epsilon}(\boldsymbol{w})]\, d\Omega \end{aligned} \tag{5.6}$$

We now take the derivative of (5.6) with respect to β:

$$\begin{aligned} \frac{dC(\mathbf{u}^* + \beta \mathbf{w})}{d\beta} = -\int_\Omega [I_1(\boldsymbol{x}) - I_2(\boldsymbol{x} + \mathbf{u}^* + \beta \mathbf{w})] \frac{\partial I_2(\boldsymbol{x} + \mathbf{u}^* + \beta \mathbf{w})}{\partial x_i} w_i \; d\Omega \\ + \frac{\alpha}{2}\int_\Omega \boldsymbol{\epsilon}(\boldsymbol{w}) : [\boldsymbol{\epsilon}(\mathbf{u}^*) + \beta \boldsymbol{\epsilon}(\boldsymbol{w})] + [\boldsymbol{\epsilon}(\mathbf{u}^*) + \beta \boldsymbol{\epsilon}(\boldsymbol{w})] : \boldsymbol{\epsilon}(\boldsymbol{w})\, d\Omega \end{aligned} \tag{5.7}$$

In equation (5.7), we introduce another aspect of index notation. When repeating index variables, we sum over them:

$$\frac{\partial \mathrm{I}_2(\boldsymbol{x}+\mathbf{u}^*+\beta\mathbf{w})}{\partial \mathrm{x_i}}\mathrm{w_i} = \frac{\partial \mathrm{I}_2(\boldsymbol{x}+\mathbf{u}^*+\beta\mathbf{w})}{\partial \mathrm{x}_1}\mathrm{w}_1 + \frac{\partial \mathrm{I}_2(\boldsymbol{x}+\mathbf{u}^*+\beta\mathbf{w})}{\partial \mathrm{x}_2}\mathrm{w}_2 + \frac{\partial \mathrm{I}_2(\boldsymbol{x}+\mathbf{u}^*+\beta\mathbf{w})}{\partial \mathrm{x}_3}\mathrm{w}_3 \tag{5.8}$$

Our next step is to set $\beta = 0$:

$$\left.\frac{\mathrm{dC}(\mathbf{u}^*+\beta\mathbf{w})}{\mathrm{d}\beta}\right|_{\beta=\mathbf{0}} = -\int_\Omega [I_1(\boldsymbol{x}) - I_2(\boldsymbol{x}+\mathbf{u}^*)]\frac{\partial I_2(\boldsymbol{x}+\mathbf{u}^*)}{\partial x_i}w_i \; d\Omega + \alpha\int_\Omega \boldsymbol{\epsilon}(\mathbf{u}^*):\boldsymbol{\epsilon}(\boldsymbol{w})\, d\Omega \tag{5.9}$$

We can simplify the strain multiplication in the second term through the following process:

$$\boldsymbol{\epsilon}(\boldsymbol{u}^*):\boldsymbol{\epsilon}(\boldsymbol{w}) = \epsilon_{ij}(\boldsymbol{u}^*)\epsilon_{ij}(\boldsymbol{w}) \tag{5.10}$$

$$= \frac{1}{2}(\partial_\mathrm{i}\mathrm{u}_\mathrm{j}^* + \partial_\mathrm{j}\mathrm{u}_\mathrm{i}^*)\;\frac{1}{2}(\partial_\mathrm{i}\mathrm{w}_\mathrm{j} + \partial_\mathrm{j}\mathrm{w}_\mathrm{i}) \tag{5.11}$$

$$= \frac{1}{4}\left[\left(\partial_\mathrm{i}\mathrm{u}_\mathrm{j}^* + \partial_\mathrm{j}\mathrm{u}_\mathrm{i}^*\right)\partial_\mathrm{i}\mathrm{w}_\mathrm{j} + \left(\partial_\mathrm{i}\mathrm{u}_\mathrm{j}^* + \partial_\mathrm{j}\mathrm{u}_\mathrm{i}^*\right)\partial_\mathrm{j}\mathrm{w}_\mathrm{i}\right] \tag{5.12}$$

$$= \frac{1}{4}\left[\left(\partial_\mathrm{i}\mathrm{u}_\mathrm{j}^* + \partial_\mathrm{j}\mathrm{u}_\mathrm{i}^*\right)\partial_\mathrm{i}\mathrm{w}_\mathrm{j} + \left(\partial_\mathrm{j}\mathrm{u}_\mathrm{i}^* + \partial_\mathrm{i}\mathrm{u}_\mathrm{j}^*\right)\partial_\mathrm{i}\mathrm{w}_\mathrm{j}\right] \tag{5.13}$$

$$= \frac{1}{2}\left(\partial_\mathrm{i}\mathrm{u}_\mathrm{j}^* + \partial_\mathrm{j}\mathrm{u}_\mathrm{i}^*\right)\partial_\mathrm{i}\mathrm{w}_\mathrm{j} \tag{5.14}$$

$$= \epsilon_{\mathrm{ij}}(\mathbf{u}^*)\,\partial_\mathrm{i}\mathrm{w}_\mathrm{j} \tag{5.15}$$

$$= \boldsymbol{\epsilon}(\boldsymbol{u}^*):\boldsymbol{\nabla}(\boldsymbol{w}) \tag{5.16}$$

Using (5.16), we rewrite (5.9):

$$\left.\frac{\mathrm{dC}(\mathbf{u}^*+\beta\mathbf{w})}{\mathrm{d}\beta}\right|_{\beta=\mathbf{0}} = -\int_\Omega [I_1(\boldsymbol{x}) - I_2(\boldsymbol{x}+\mathbf{u}^*)]\frac{\partial(\boldsymbol{x}+\mathbf{u}^*)}{\partial x_i}w_i \; d\Omega + \alpha\int_\Omega = -\boldsymbol{\epsilon}(\boldsymbol{u}^*):\boldsymbol{\nabla}(\boldsymbol{w})\, d\Omega \tag{5.17}$$

We then drop the * on $\boldsymbol{u}^*$, thus replacing $\boldsymbol{u}^*$ with $\boldsymbol{u}$, defining $\boldsymbol{u} = \boldsymbol{u^0} + \delta\boldsymbol{u}$ ($\boldsymbol{u^0}$: initial guess) and setting the derivative in (5.17) equal to zero:

$$0 = -\int_\Omega \left[I_1(\boldsymbol{x}) - I_2\left(\boldsymbol{x}+\boldsymbol{u^0}+\delta\boldsymbol{u}\right)\right]\frac{\partial I_2\left(\boldsymbol{x}+\boldsymbol{u^0}+\delta\boldsymbol{u}\right)}{\partial x_i}w_i \; d\Omega \tag{5.18}$$

$$+\alpha \int_{\Omega} \frac{1}{2}\left(\partial_i u_j^0 + \partial_i \delta u_j + \partial_j u_i^0 + \partial_j \delta u_i\right)\partial_i w_j \, d\Omega$$

We approximate $I_2(\mathbf{x} + \mathbf{u^0} + \delta\mathbf{u})$ in (5.18) using Taylor series expansion as follows:

$$\mathrm{I}_2\left(\mathbf{x} + \mathbf{u^0} + \delta\mathbf{u}\right) = \mathrm{I}_2(\mathrm{x}_1 + \mathrm{u}_1^0 + \delta\mathrm{u}_1, \mathrm{x}_2 + \mathrm{u}_2^0 + \delta\mathrm{u}_2, \mathrm{x}_3 + \mathrm{u}_3^0 + \delta\mathrm{u}_3) \tag{5.19}$$

$$\approx \mathrm{I}_2(\mathrm{x}_1 + \mathrm{u}_1^0, \mathrm{x}_2 + \mathrm{u}_2^0, \mathrm{x}_3 + \mathrm{u}_3^0) + \frac{\partial \mathrm{I}_2}{\partial \mathrm{x}_1}\delta\mathrm{u}_1 + \frac{\partial \mathrm{I}_2}{\partial \mathrm{x}_2}\delta\mathrm{u}_2 + \frac{\partial \mathrm{I}_2}{\partial \mathrm{x}_3}\delta\mathrm{u}_3 \tag{5.20}$$

$$= \mathrm{I}_2\left(\mathbf{x} + \mathbf{u^0}\right) + \frac{\partial \mathrm{I}_2\left(\mathbf{x} + \mathbf{u^0}\right)}{\partial \mathrm{x}_\mathrm{j}}\delta\mathrm{u}_\mathrm{j} \tag{5.21}$$

Also, using a Gauss-Newton approximation, we approximate $\partial I_2/\partial x_i$ as:

$$\begin{aligned}\left[I_1(\boldsymbol{x}) - I_2\left(\boldsymbol{x} + \boldsymbol{u^0} + \delta\boldsymbol{u}\right)\right]\frac{\partial I_2\left(\boldsymbol{x} + \boldsymbol{u^0} + \delta\boldsymbol{u}\right)}{\partial x_i} \\ \approx \left[I_1\,(\boldsymbol{x}) - I_2\,(\boldsymbol{x} + \boldsymbol{u^0} + \delta\boldsymbol{u})\right]\frac{\partial I_2\left(\boldsymbol{x} + \boldsymbol{u^0}\right)}{\partial x_i}\end{aligned} \tag{5.22}$$

That is, we make the approximation in (5.22), because it is safe to assume that $(I_1 - I_2)$ is a small quantity, so the small change on the other factor is negligible when multiplied by a small quantity.

We now substitute (5.21) and (5.22) into (5.18):

$$\begin{aligned}0 = -\int_{\Omega}\left[I_1(\boldsymbol{x}) - \mathrm{I}_2\left(\mathbf{x} + \mathbf{u^0}\right) - \frac{\partial \mathrm{I}_2\left(\mathbf{x} + \mathbf{u^0}\right)}{\partial \mathrm{x}_\mathrm{j}}\delta\mathrm{u}_\mathrm{j}\right]\frac{\partial I_2\left(\boldsymbol{x} + \boldsymbol{u^0}\right)}{\partial x_i} w_i \; d\Omega \\ +\alpha \int_{\Omega}\left(\epsilon_{\mathrm{ij}}\left(\mathbf{u^0}\right) + \epsilon_{\mathrm{ij}}(\delta\mathbf{u})\right)\partial_i w_j \, d\Omega\end{aligned} \tag{5.23}$$

Next, we reorient (5.23) to combine terms with known $\boldsymbol{u^0}$ on the right-hand side of the equation and terms with unknown $\delta\boldsymbol{u}$ on the left-hand side (for efficient notation, we replace $I_1(\boldsymbol{x})$ with I_1 and $I_2(\mathbf{x} + \mathbf{u^0})$ with I_2):

$$\int_{\Omega}\frac{\partial I_2}{\partial x_i}\frac{\partial \mathrm{I}_2}{\partial \mathrm{x}_\mathrm{j}} w_i \delta\mathrm{u}_\mathrm{j} + \alpha\epsilon_{\mathrm{ij}}(\delta\mathbf{u})\partial_i w_j d\Omega = \int_{\Omega}[I_1 - \mathrm{I}_2]\frac{\partial I_2}{\partial x_i} w_i + \alpha\epsilon_{\mathrm{ij}}\left(\mathbf{u^0}\right)\partial_i w_j d\Omega \tag{5.24}$$

For convenience, we will adapt (5.24) such that all "w" terms have a "j" index. To do so, we switch the index variable in all appropriate terms:

$$\int_{\Omega} \frac{\partial I_2}{\partial x_i} \frac{\partial \mathrm{I}_2}{\partial \mathrm{x_j}} w_j \delta \mathrm{u_i} + \alpha \epsilon_{\mathrm{ij}}(\delta \mathbf{u}) \partial_i w_j d\Omega = \int_{\Omega} [I_1 - \mathrm{I}_2] \frac{\partial I_2}{\partial x_j} w_j - \alpha \epsilon_{\mathrm{ij}}(\mathbf{u^0}) \partial_i w_j d\Omega \tag{5.25}$$

To discretize and allow us to use shape functions N^A or N^B, let

$$\delta \mathrm{u_i} = \sum_{\mathrm{A=1}}^{\mathrm{n_{nodes}}} \delta \mathrm{u_i^A N^A} = \delta \mathrm{u_i^A N^A} \tag{5.26}$$

$$\mathrm{w_j} = \sum_{\mathrm{B=1}}^{\mathrm{n_{nodes}}} \mathrm{w_j^B N^B} = \mathrm{w_j^B N^B} \tag{5.27}$$

In the two equations immediately above, we removed the summation sign since we will follow the summation convention for the superscripts as well. The repeated superscripts indicate that we apply the summation over each node in the finite element mesh detailed in section 3.3.

As an aside, equations (5.26) and (5.27) present a branching point for DVC, as introduced in chapter 1, because algorithms differ in what shape functions are used. In our case, we use eight-nodal, hexahedral elements with trilinear shape functions, which are analogous to the linear in 1D N_A and N_B presented in chapter 1.

Moving forward, we substitute (5.26) and (5.27) into (5.25):

$$\begin{aligned} \int_{\Omega} \frac{\partial I_2}{\partial x_i} \frac{\partial \mathrm{I}_2}{\partial \mathrm{x_j}} \mathrm{w_j^B N^B} \delta \mathrm{u_i^A N^A} + \alpha \epsilon_{\mathrm{ij}}(\delta \mathbf{u^A N^A}) \partial_i \mathrm{w_j^B N^B} d\Omega \\ = \int_{\Omega} [I_1 - \mathrm{I}_2] \frac{\partial I_2}{\partial x_j} \mathrm{w_j^B N^B} - \alpha \epsilon_{\mathrm{ij}}(\mathbf{u^0}) \partial_i \mathrm{w_j^B N^B} d\Omega \end{aligned} \tag{5.28}$$

We can simplify (5.28) further by first recognizing that there is a w_j^B *in every term.* Therefore, we factor it out. Recognizing that w_j^B is arbitrary leads to 3 equations for each node, one for each w_j^B:

$$\int_\Omega \frac{\partial I_2}{\partial x_i}\frac{\partial I_2}{\partial x_j} N^B \delta u_i^A N^A + \alpha \epsilon_{ij}\left(\delta \mathbf{u^A N^A}\right)\partial_i N^B d\Omega = \int_\Omega [I_1 - I_2]\frac{\partial I_2}{\partial x_j} N^B - \alpha \epsilon_{ij}\left(\mathbf{u^0}\right)\partial_i N^B d\Omega \tag{5.29}$$

To simplify further, we seek to factor out the δu_i^A from the left-hand side. To do so, let us expand $\epsilon_{ij}(\delta \mathbf{u^A N^A})$ *from* the second term and adjust the indexing variable:

$$\epsilon_{ij}\left(\delta \mathbf{u^A N^A}\right)\partial_i N^B = \frac{1}{2}(\partial_i \delta u_j^A N^A + \partial_j \delta u_i^A N^A)\partial_i N^B \tag{5.30}$$

$$= \frac{1}{2}\left(\delta_{ij}\,\partial_k \delta u_i^A N^A\right)\partial_k N^B + \left(\partial_j \delta u_i^A N^A\right)\partial_i N^B \tag{5.31}$$

In (5.31) we switch the index of "∂" in the first term from "i" to "k" to avoid summation and we introduce Kronecker delta, δ_{ij}, which is defined as:

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{5.32}$$

We now substitute (5.31) into (5.29) to obtain:

$$\int_\Omega \frac{\partial I_2}{\partial x_i}\frac{\partial I_2}{\partial x_j} N^B \delta u_i^A N^A + \frac{\alpha}{2}\left(\delta_{ij}\,\partial_k N^A \partial_k N^B + \partial_j N^A \partial_i N^B\right)\delta u_i^A d\Omega = \int_\Omega [I_1 - I_2]\frac{\partial I_2}{\partial x_j} N^B - \alpha \epsilon_{ij}\left(\mathbf{u^0}\right)\partial_i N^B d\Omega \tag{5.33}$$

We now factor out the δu_i^A from the left-hand side of (5.33)*:*

$$\left[\int_\Omega \frac{\partial I_2}{\partial x_i}\frac{\partial I_2}{\partial x_j} N^B N^A + \frac{\alpha}{2}\left(\delta_{ij}\,\partial_k N^A \partial_k N^B + \partial_j N^A \partial_i N^B\right)d\Omega\right]\delta u_i^A = \int_\Omega [I_1 - I_2]\frac{\partial I_2}{\partial x_j} N^B - \alpha \epsilon_{ij}\left(\mathbf{u^0}\right)\partial_i N^B d\Omega \tag{5.34}$$

This now enables us to define:

$$\mathrm{K}_{\mathrm{ji}}^{\mathrm{BA}} = \int_{\Omega} \frac{\partial I_2}{\partial x_i} \frac{\partial \mathrm{I}_2}{\partial \mathrm{x}_\mathrm{j}} \mathrm{N}^{\mathrm{B}} \mathrm{N}^{\mathrm{A}} + \frac{\alpha}{2} \left(\delta_{\mathrm{ij}} \, \partial_{\mathrm{k}} \mathrm{N}^{\mathrm{A}} \partial_k \mathrm{N}^{\mathrm{B}} \; + \partial_{\mathrm{j}} \mathrm{N}^{\mathrm{A}} \partial_i \mathrm{N}^{\mathrm{B}} \right) d\Omega \tag{5.35}$$

$$\mathrm{F}_{\mathrm{j}}^{\mathrm{B}} = \int_{\Omega} [I_1 - \mathrm{I}_2] \frac{\partial I_2}{\partial x_j} \mathrm{N}^{\mathrm{B}} - \alpha \epsilon_{\mathrm{ij}} \left(\mathbf{u^0} \right) \partial_i \mathrm{N}^{\mathrm{B}} d\Omega \tag{5.36}$$

$\mathrm{K}_{\mathrm{ji}}^{\mathrm{BA}}$ is the stiffness matrix and $\mathrm{F}_{\mathrm{j}}^{\mathrm{B}}$ is the force vector, similar to those presented in chapter 1. Similarly, we solve for the nodal displacements using the following relationship:

$$\delta \mathbf{u} = [\mathbf{K}]^{-1} \mathbf{F} \tag{5.37}$$

In (5.37), $\delta \mathbf{u}$ is a column matrix with entries $\delta \mathrm{u}_{\mathrm{i}}^{\mathrm{A}}$, $\mathbf{K}$ is a square matrix with entries $\mathrm{K}_{\mathrm{ji}}^{\mathrm{BA}}$ and finally, $\mathbf{F}$ is another column vector with entries $\mathrm{F}_{\mathrm{j}}^{\mathrm{B}}$ corresponding to each mesh node. Note that we create a single index with a length of three times the number of nodes so that $J = 3(B-1) + j$ and $F_J = F_j^B$. A similar approach is used for $\mathbf{K}$.

5.2. Appropriate Choice of Initial Guess

Our DVC algorithm is iterative and so requires an initial guess. If the initial guess is too far from the correct solution, then the algorithm will converge to the wrong minimum. In this section, we describe how to obtain an initial guess that will converge, and we also try to give an intuitive sense of what determines "close enough".

5.2.1. Defining an appropriate initial guess in 3D

In 3D, our initial guess is a 3-by-n_{nodes} matrix; that is, we provide the initial guess as x-, y-, and z-displacements for each mesh node.

Example 1: Exploration of an appropriate initial guess in 3D.

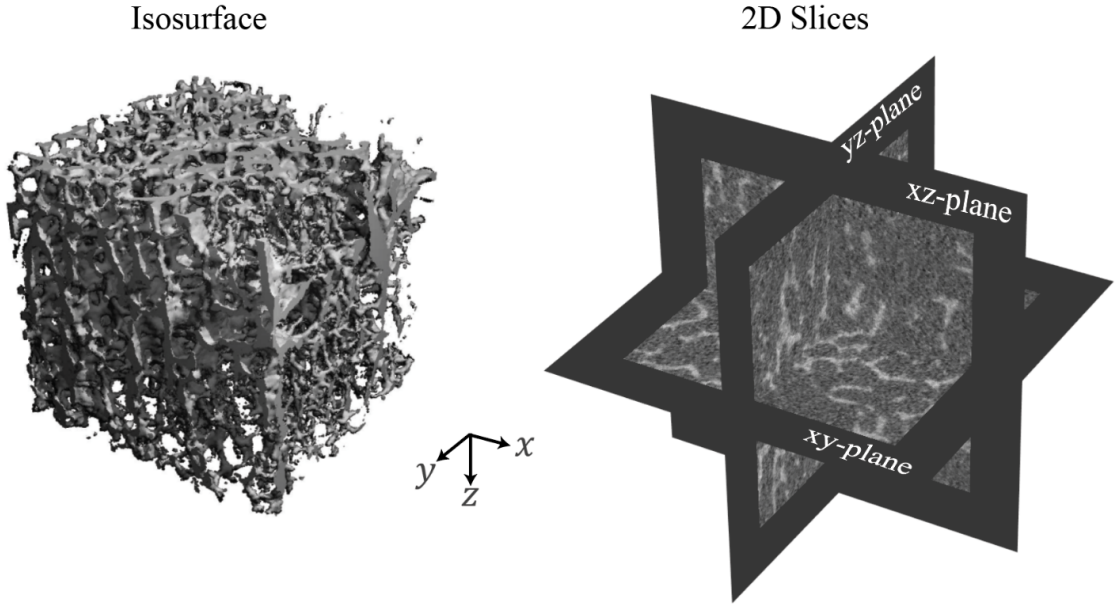


Figure 5.1: Cubic region of trabecular bone contoured from a μCT scan of a human vertebra. We depict this region with both an isosurface and set of 2D slices.

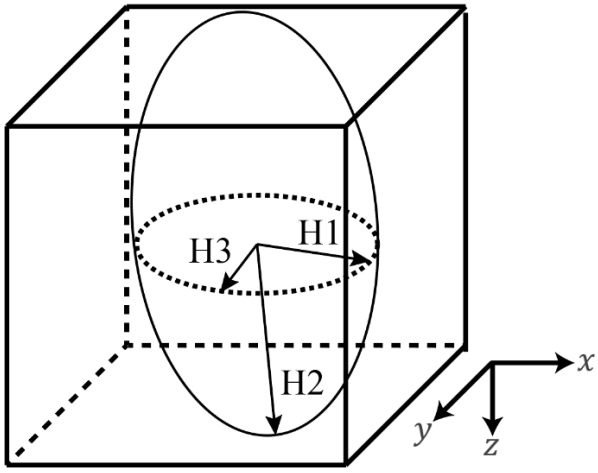
To explore what is "close enough" for an initial guess in 3D, we will use an approach similar to that used in section 1.2.5, where we apply to an image a rigid translation of nodal displacement along a single axis. For this example, we will use a cubic volume of trabecular bone excised from an image of a full vertebra (see Figure 5.1). We measured the morphology of this bone specimen using Scanco evaluation software where we applied a threshold value of 263.7 mg HA/ccm. Results are located in Table 5.1 and Figure 5.2. The results that stand out are trabecular thickness (Tb.Th), trabecular separation (Tb.Sp) and structural anisotropy.

To understand the significance of the Tb.Th and Tb.Sp results, we recall that the image of the cubic bone sample is a 3D intensity distribution. The intensity is maximal in the trabecula. Hence, the trabecula intensity peaks are analogous to the 1D intensity peaks used

BONE VOLUME FRACTION (BV/TV)	0.1476
CONNECTIVITY DENSITY (CONN. D.)	0.1115 (1/vox)
TRABECULAR THICKNESS (TB.TH)	5.6432 (vox)
TRABECULAR SEPARATION (TB.SP)	21.5919 (vox)

Table 5.1: Bone morphology measures of the cubic bone sample.

Structural Anisotropy



Eigenvalues	Eigenvectors
$\|H1\| = 0.9271$	$H1 = [\ 0.8905 \quad 0.2521 \quad 0.0534]$
$\|H2\| = 1.1731$	$H2 = [\ 0.0902 \quad -0.0713 \quad -1.1674]$
$\|H3\| = 0.9979$	$H3 = [-0.2665 \quad 0.9584 \quad -0.0792]$

Figure 5.2: Structural anisotropy of the cubic bone sample. The representative ellipsoid above has radii with directions H1, H2, and H3 and lengths |H1|, |H2|, and |H3|.

in section 1.2 (see Figure 5.2). We saw in the 1D case (in section 1.2.7) that the farther the intensity peaks in the two images are from each other, the more likely we are to require a large number of iterations to converge, and that may be to the wrong displacement field (*i.e.*, local minimum). The large peaks in Figure 5.3 represent the key intensity "features" of our image, which in this case is a single trabecula, or trabecular strut. These struts have an average thickness of Tb.Th = 5.6 voxels. This means that the representative intensity peak has that width on average. The more that the intensity peaks of each image overlap, the faster our convergence will be. Also, according to our Tb.Sp measure, these struts are typically spaced Tb.Sp = 21.6 voxels apart. Thus, we would expect that a displacement shift, u_s, greater than roughly half the spacing would result in convergence on a local minimum (see section 1.2.7). In other words, we would align the intensity peaks from one image to the wrong peaks of the other.

Structural anisotropy is a measure of the directional distribution of material. The outputs of an anisotropy analysis indicate whether the trabecular bone aligns more along one specific direction. The anisotropy results, shown in Figure 5.2, tell us that there is a preferential alignment of bone along the z-axis, and approximately isotropic distribution within the xy-plane. We take advantage of the similarity between the x and y directions by choosing just two directions for our shift study: along the y- and z-axes.

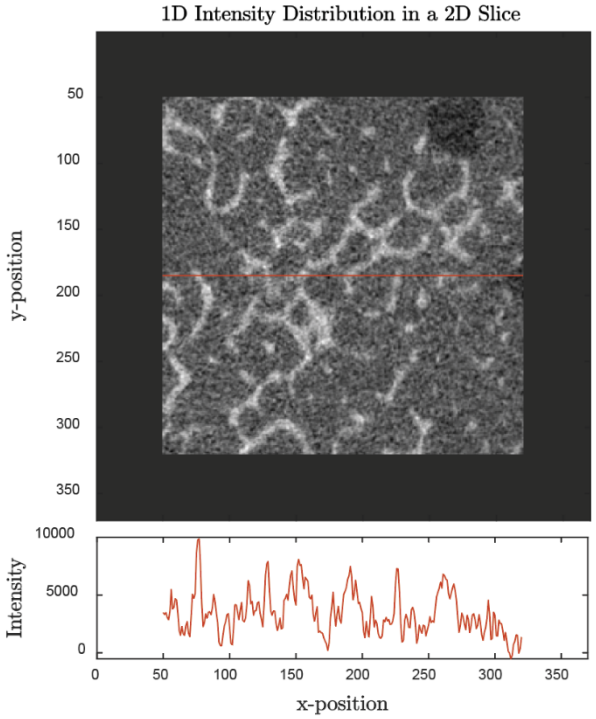


Figure 5.3: 1D intensity distribution of a 2D slice. The red line plotted below represents the intensity readings followed by the red line in the slice above.

Therefore, we start our example with a rigid displacement of thirty voxels applied individually along the y- and z-axes (see Figure 5.4), thus creating two test cases for our study. In the input file (data.in), we manually edit the initial guess of nodal displacement. This allows us to vary how distant our initial guess is from the true displacement. We label this distance u_s, which is a value we increase to simulate an increasingly poor initial guess. Our convergence criterion is that $RMS_{\delta u}$ (as defined in section 1.2.4) must reach a value below 0.001 voxels. However, if DVC fails to reach convergence in less than 31 iterations, we stop the algorithm.

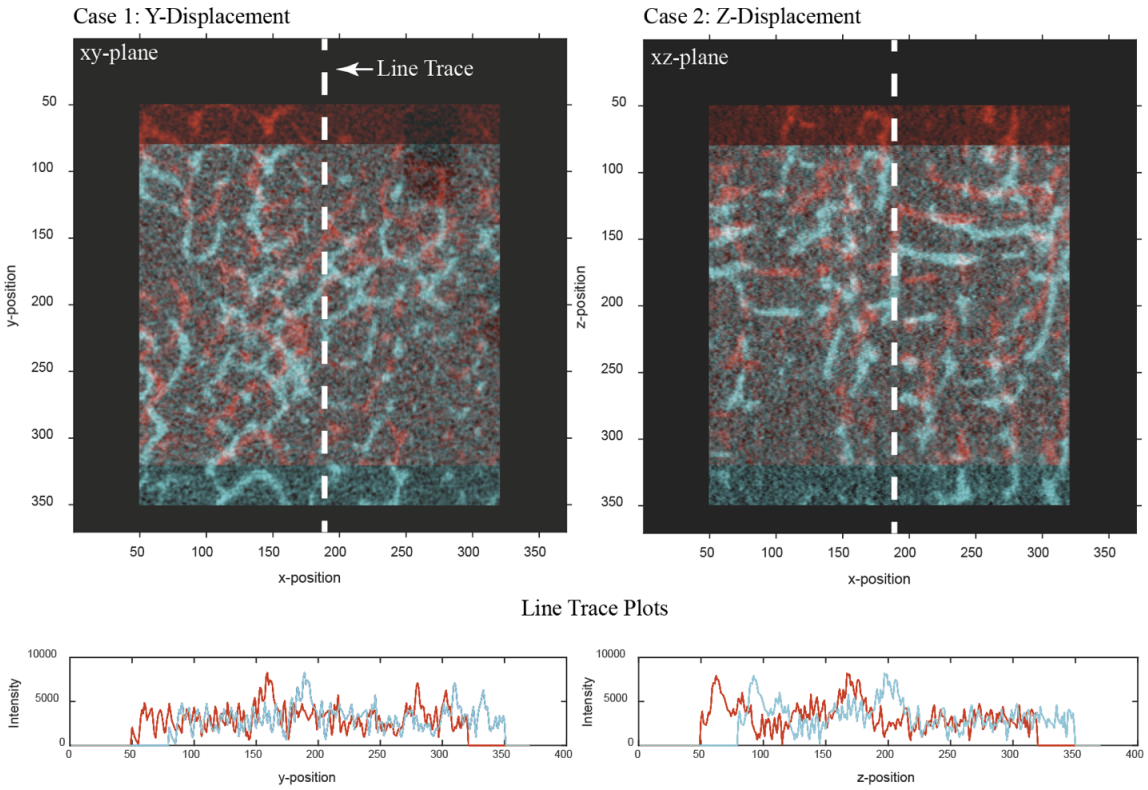


Figure 5.4: Both rigid displacement cases we use to explore 3D initial guess. For each case, we applied a rigid displacement of thirty voxels (+) along the y-axis for the first case and the z-axis for the second. The undeformed image is depicted in red and the deformed in cyan.

Our initial exploration of u_s (see Figure 5.5) showed that at a certain threshold ($u_s = 8$ for y-displacement and $u_s = 6$ for z-displacement), the decrease in $RMS_{\delta u}$ slows down and convergence is not reached before the user-defined, iteration limit. To clarify further, we compare $RMS_{\delta u}$ to iterations before convergence (see Figure 5.6), where we only plot the initial guesses that appear to either approach or arrive at convergence. Doing so reveals that, for these two displacement cases and the initial guesses explored, we are following the same path to convergence. There is an initial, very slow movement towards the right displacement shift. Once within a certain distance of the right answer, there is a rapid drop in $RMS_{\delta u}$. The initial guess we choose determines our starting point along that path. Also,

we noticed a cyclic behavior along the initial portion of this convergence path. This is possibly due to the cyclic error inherent in our method of image interpolation. Interpolation error drops down to zero as we make integer shifts in displacement. Finally, returning to Figure 5.5, there appears to be a third category of initial guesses ($u_s \geq 15$) for which convergence will fail. Convergence failure is defined as reaching the $RMS_{\delta u}$ threshold without actually improving image matching (*i.e.*, without achieving a low mismatch value). This behavior means we have settled on a local minimum; we have gone down the wrong path.

Figure 5.7 and Figure 5.8 show qualitative results of the y- and z-displacement cases, respectively. In this figure, we display the highest u_s where convergence was just about to occur prior to cutoff, the next u_s value, and a u_s where convergence on a local minimum occurs. For the second of these three cases, it is clear that we are approaching the true displacement field.

The requirement that our initial guess must be within six (z-displacement) to eight (y-displacement) voxels from the true solution for quick convergence to occur and, worse, that if we are only 15 voxels away we will fail to converge on the true displacement, means that there is little room for error. This range coincides with the trabecular thickness we reported above. Once our image misalignment distance is greater than 5.6 voxel Tb.Th, there is a decreasing peak overlap, which slows down convergence. We observed this when $u_s = 9 - 11$ for y-displacement and $u_s = 7 - 9$ for z-displacement. We can also take into account trabecular separation where a peak distance greater than half that spacing

($(21.6\ vox)/2 = 10.8\ vox$) results in convergence on a local minimum. Such behavior is seen when $u_s \geq 15$ in both y- and z-displacement. To put it plainly, as we increase the magnitude of u_s past Tb.Th, we slow down convergence and past half the Tb.Sp, convergence will both slow down *and* arrive at an inaccurate solution. Finally, the difference in convergence behavior between both cases (y- and z-displacement) is likely attributed to the difference of trabecular bone alignment as shown in the anisotropy results. In other words, while we report an average Tb.Th and Tb.Sp for our cubic sample, similar measures individually averaged across each axis would likely show a significant difference. The difference in spacing between trabeculae between y- and z-directions is made evident with a qualitative review of the images in Figure 5.4.

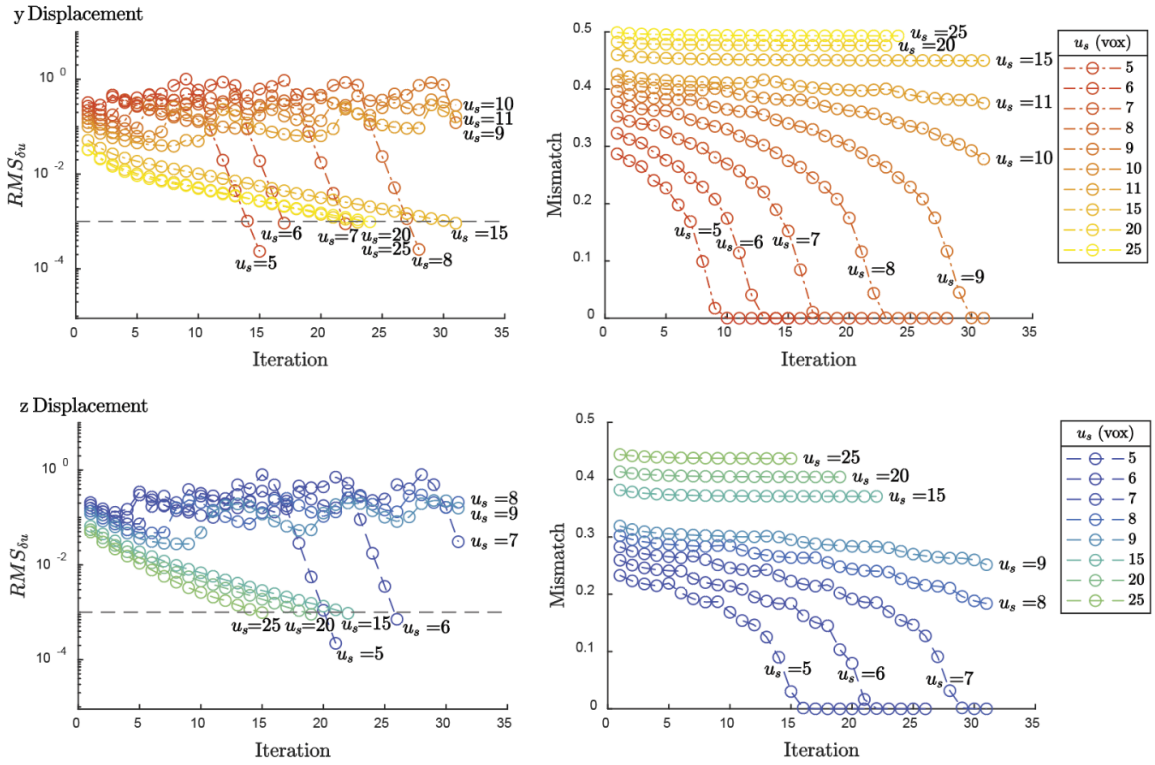


Figure 5.5: Convergence behavior when varying u_s for both y-displacement (top) and z-displacement (bottom) are depicted in the left plots. To the right, we have plots of mismatch values at each iteration. The z-displacement case failed at a lower u_s than the y-displacement case. In both cases, the highest u_s with good image matching failed to reach the $RMS_{\delta u} = 1e-3$ threshold suggesting that they would converge if given more iterations.

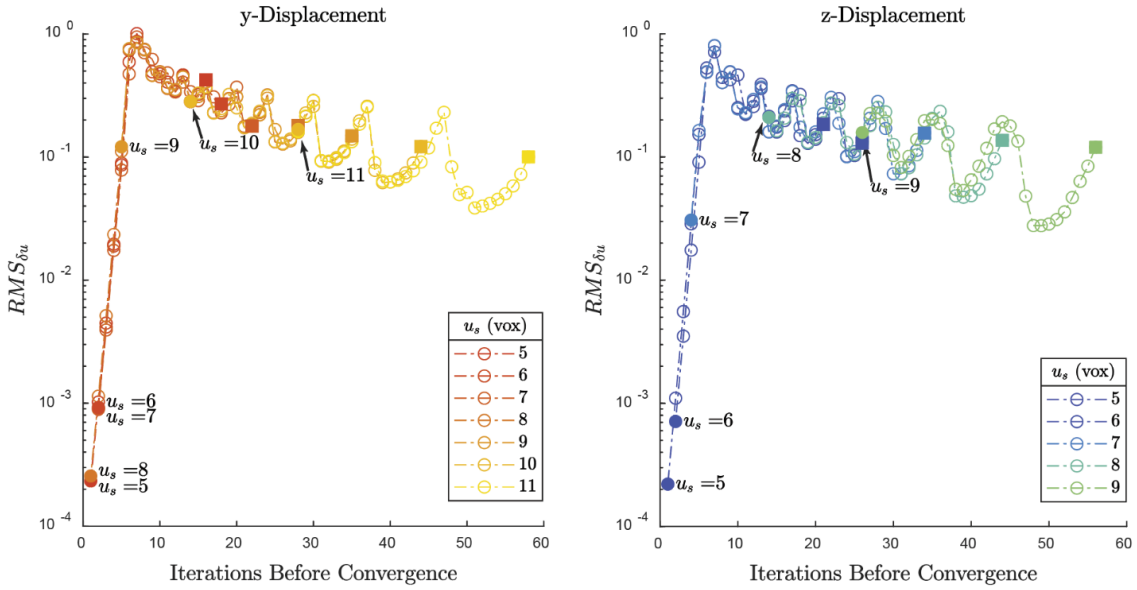


Figure 5.6: Convergence plots where we instead plot iterations before convergence in the horizontal axis. Doing so allows us to see how, for each these two cases, and when starting at different initial guess values, we are essentially solving the same problem but are choosing different starting points. The filled square and circle markers indicate the starting and ending iterations for each u_s, respectively.

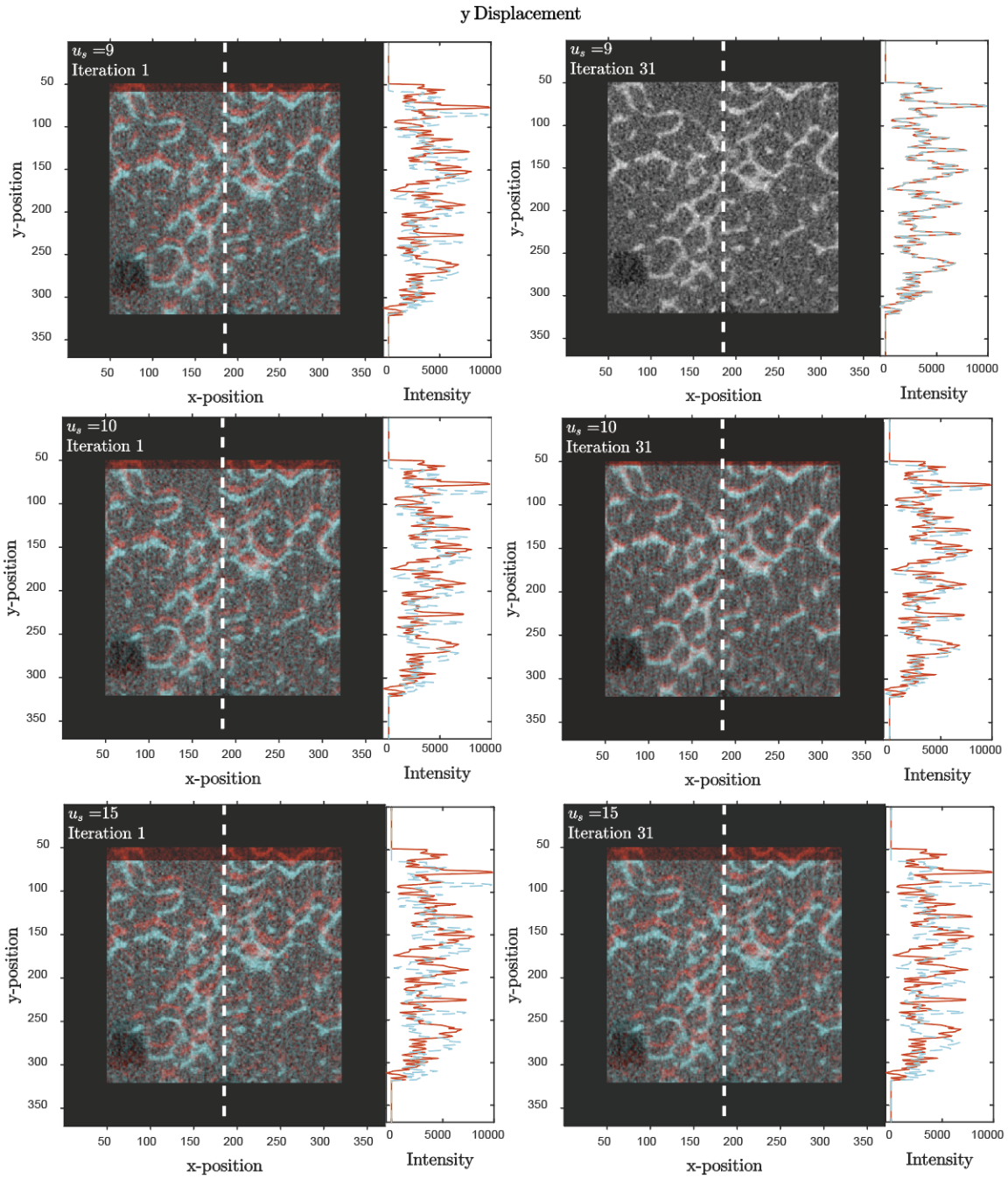


Figure 5.7: Images displaced by the DVC-derived, y-displacement field. We look at three u_s values per case. The first is where convergence did not occur before reaching the $RMS_{\delta u}$ threshold, but a mismatch value of zero was achieved. The second is immediately following the first. Note that in the second u_s, the images appear to be approaching the correct displacement field. The third is where we converge on a local minimum.

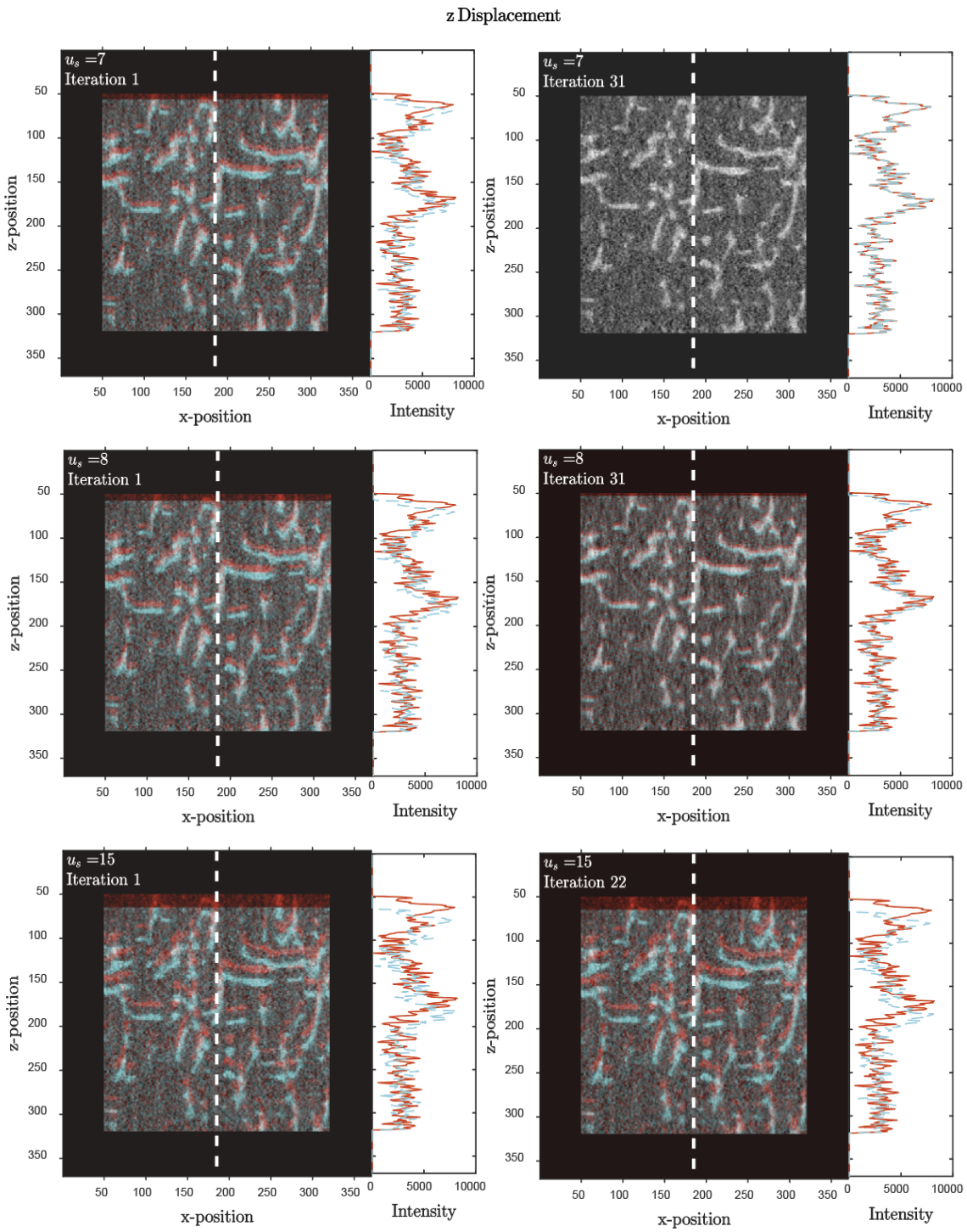


Figure 5.8: Images displaced by the DVC-derived displacement field. We look at two u_s values per case. The first is where convergence did not occur before reaching the $RMS_{\delta u}$ threshold, but a mismatch value of zero was achieved. The second is immediately following the first. Note that in the second u_s, the images appear to be approaching the correct

5.2.2. Downsampling to address stringent initial guess requirements

Downsampling alleviates the stringent initial guess requirements demonstrated in the previous example.

Example 2: Exploration of an appropriate initial guess in 3D when downsampling.

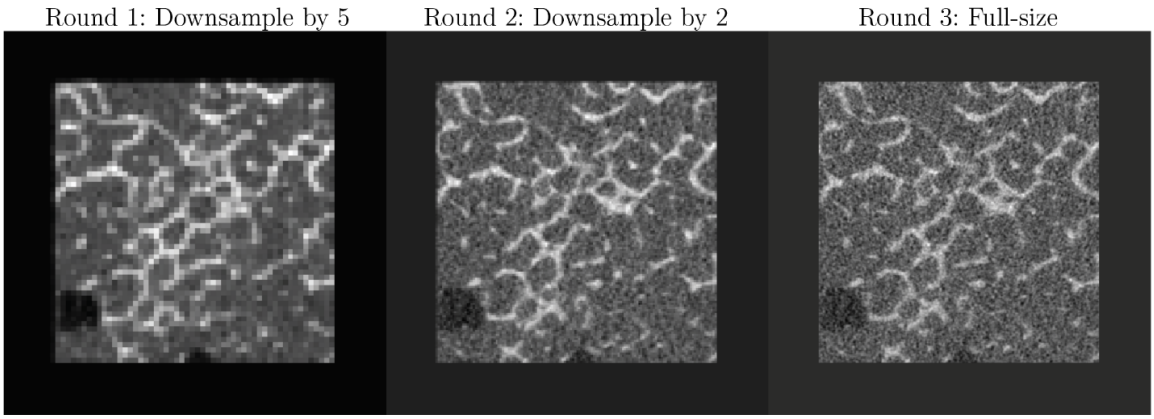


Figure 5.9: Slice taken from the cubic sample as it goes through each round in our pyramidal approach. The DVC-calculate displacement at the end of each round is used as the initial guess for the following round. Isotropic voxel size for each round was 0.185mm for downsample by 5, 0.074 for downsample by 2, and 0.037 for full-size.

To first see the impact of downsampling on the effectiveness of initial guess, we explored three different images scales: downsampled by five, downsampled by two, and full-sized (shown in Figure 5.9) for both the y- and z-displacement cases (Figure 5.10 and Figure 5.11, respectively). Note that we scale u_s by the same downsample factor used for the image. For example, $u_s = 10$ (full-size) voxels was scaled down to $u_s = 2$ (downsampled) voxels when downsampling the image by a factor of five and $u_s = 5$ (downsampled) voxels when downsampling by a factor of two. To simplify comparison across image scales in Figure 5.10 through Figure 5.16, all values of u_s stated in this figure correspond to full-size voxels (e.g. "$u_s = 10$" means 10 full-size voxels for the full-size cases, 5 downsampled voxels for downsampling by a factor of 2, and 2 downsampled

voxels for downsampling by a factor of 5). In both displacement cases, when downsampling, we not only saw faster convergence, but more importantly, we were able to converge on a correct solution for higher u_s values, i.e. worse initial guesses. The y-displacement case, however, did show that there still exists a limit to how poor an initial guess can be ($u_s = 20$ (full-size) voxels) even after downsampling by as much as a factor of five. Perhaps a higher downsampling factor would have enabled convergence.

In our DVC algorithm, we follow the pyramidal procedure described in section 3.5, (see Figure 5.9 for representative slices at each step of the procedure). In Figure 5.12 and Figure 5.13, we present $RMS_{\delta u}$ convergence and mismatch plots for both y- and z-displacement cases using our pyramidal DVC starting with u_s values of ten, which we increase until we fail to reach zero-mismatch. For both cases, u_s is allowed to be higher than what we see in field. To demonstrate the behavior of pyramidal DVC, we first focus on the case of y-displacement with $u_s = 10$ (full-size) voxels. This same initial guess was used in both full resolution (previous section) and pyramidal DVC. When comparing performance (see Figure 5.14), we see that a downsampled image pair requires many fewer iterations to the previous section (compare to Figure 5.5), and still converge on the correct displacement converge to the right solution. Furthermore, each iteration, for the downsampled images, is completed much faster than for the full size images (see section 3.5.1). In fact, while the full resolution approach took fifteen minutes to converge to an incomplete solution, it only took the pyramidal approach one minute to converge to the correct one.

Next, we look at the case of z-displacement with $u_s = 40$ (full-size) voxels (see Figure

5.15) which presents a situation where convergence did not occur until the second round. In the first round, we see that DVC brought the images toward the correct displacement field. This set up the second round for success by essentially correcting the poor initial guess we started with.

For the y-displacement case, a noteworthy phenomenon occurs when u_s=18 (full-size) voxels (see Figure 5.16). Here, in the first round of our pyramidal DVC, convergence occurs in few iterations, while the mismatch value clearly has not only not reached zero, but is trending away. Again, this is characteristic of convergence on a local minimum. This could have possibly been avoided with our pyramidal approach had we added one more round before the first where we downsample by a higher factor. For future performance improvement, we would recommend this additional step as it would add minimal computational cost while greatly discouraging convergence on local minima.

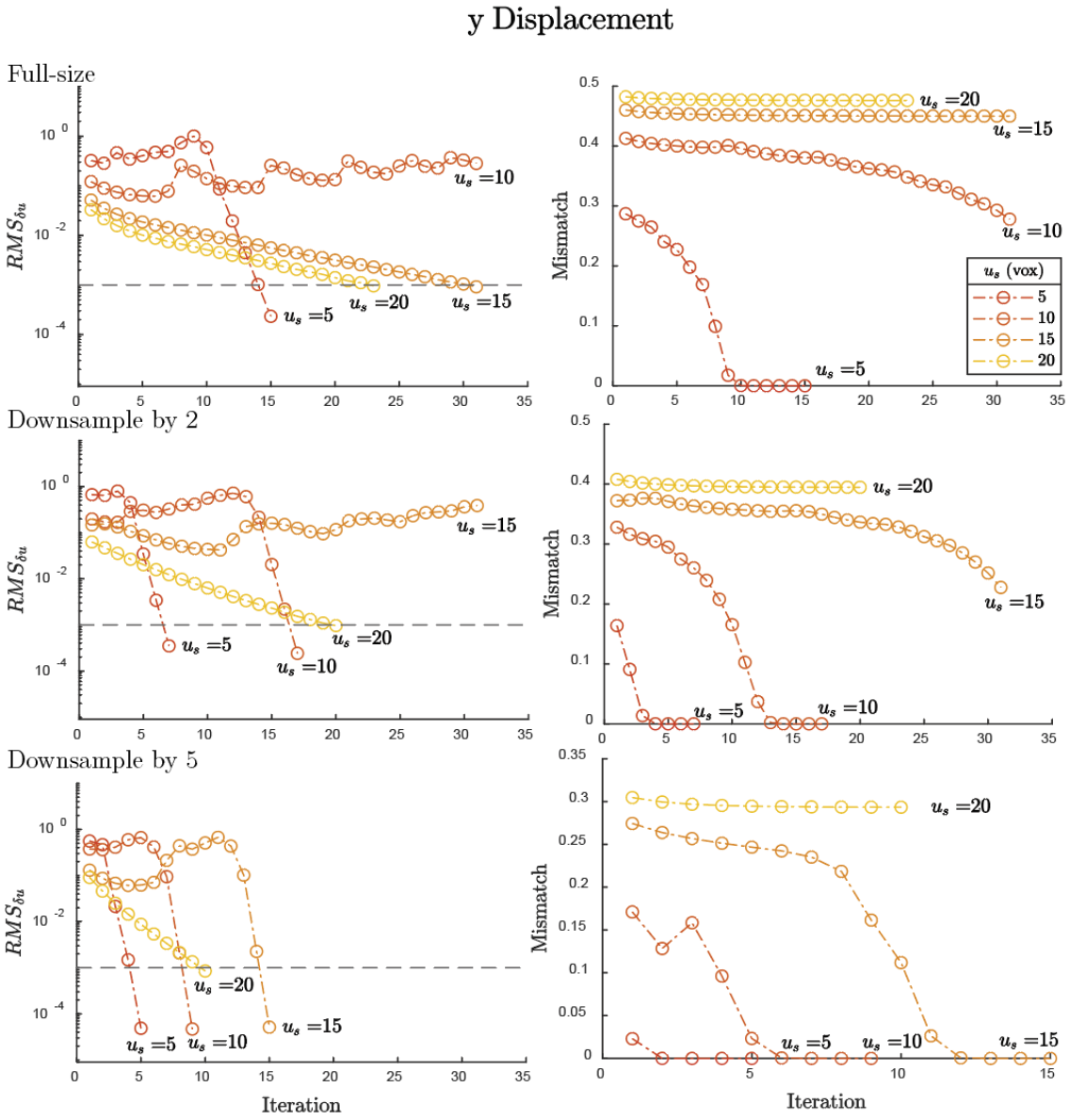


Figure 5.10: Convergence behavior measured for the y-displacement case with $RMS_{\delta u}$ (left) and mismatch (right) for the full-sized (top row), downsampled by two (middle row), and downsampled by five (bottom row) images. Isotropic voxel size for each scale was 0.185mm for downsample by 5, 0.074 for downsample by 2, and 0.037 for full-size.

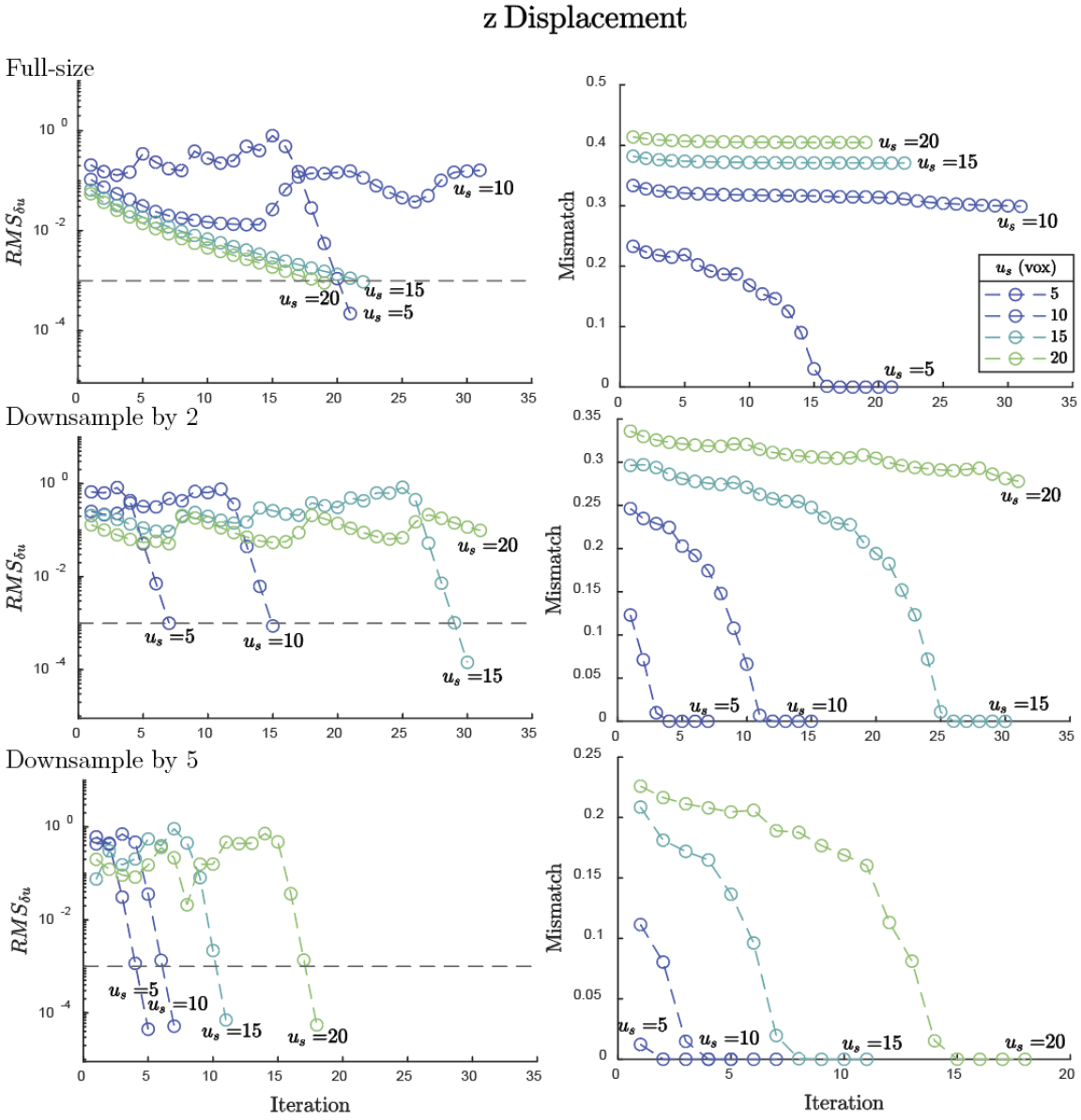


Figure 5.11: Convergence behavior measured for the z-displacement case with $RMS_{\delta u}$ (left) and mismatch (right) for the full-sized (top row), downsampled by two (middle row), and downsampled by five (bottom row) images. Isotropic voxel size for each scale was 0.185mm for downsample by 5, 0.074 for downsample by 2, and 0.037 for full-size.

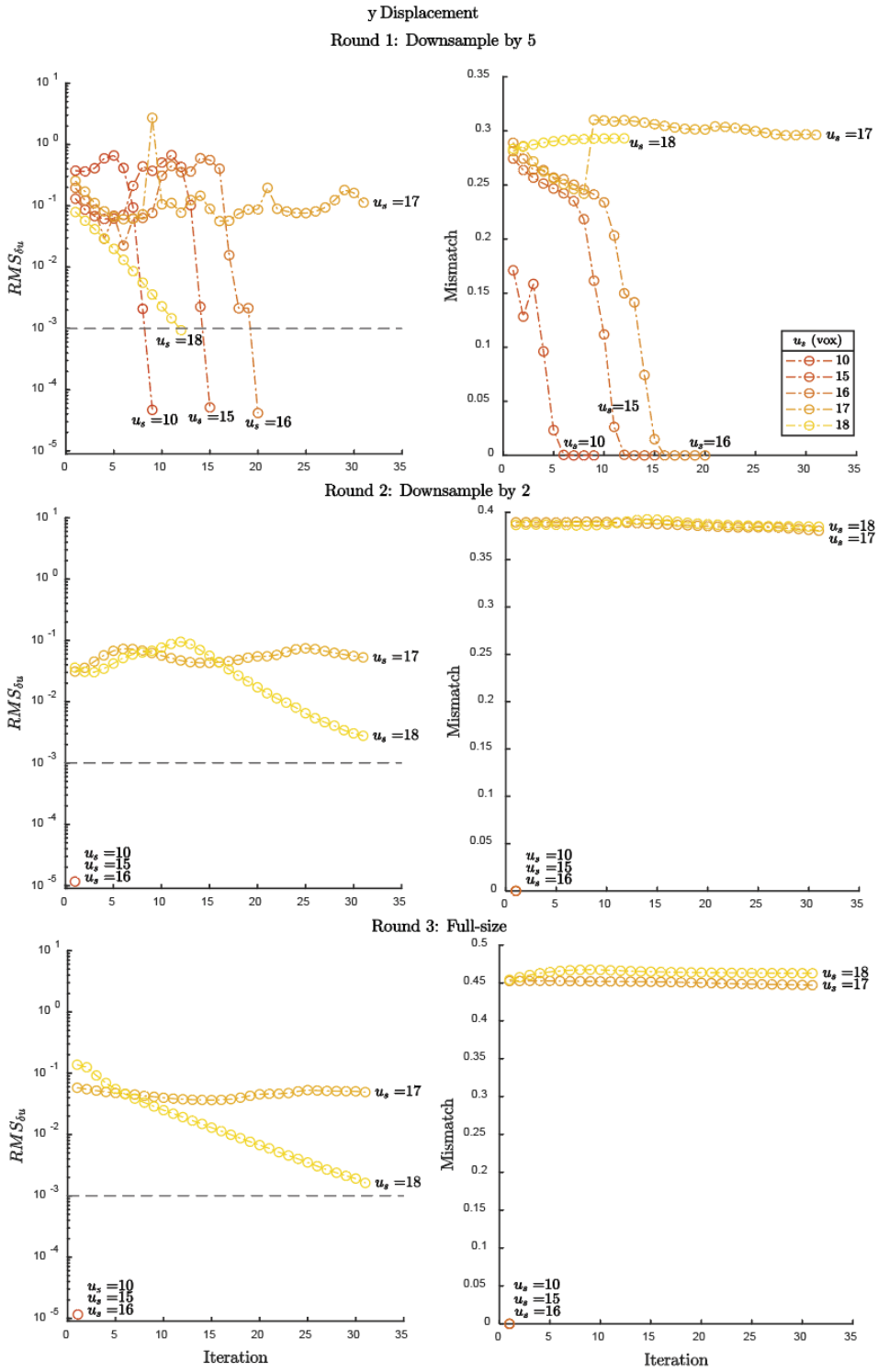


Figure 5.12: $RMS_{\delta u}$ and mismatch plots at each round for the y-displacement case. The u_s values shown are relative to the full-size image and scaled down for the earlier rounds. In fact, the u_s = 10 case shown here is comparable to the same shown in Figure 5.5. Isotropic voxel size for each scale was 0.185mm for downsample by 5, 0.074 for downsample by 2, and 0.037 for full-size.

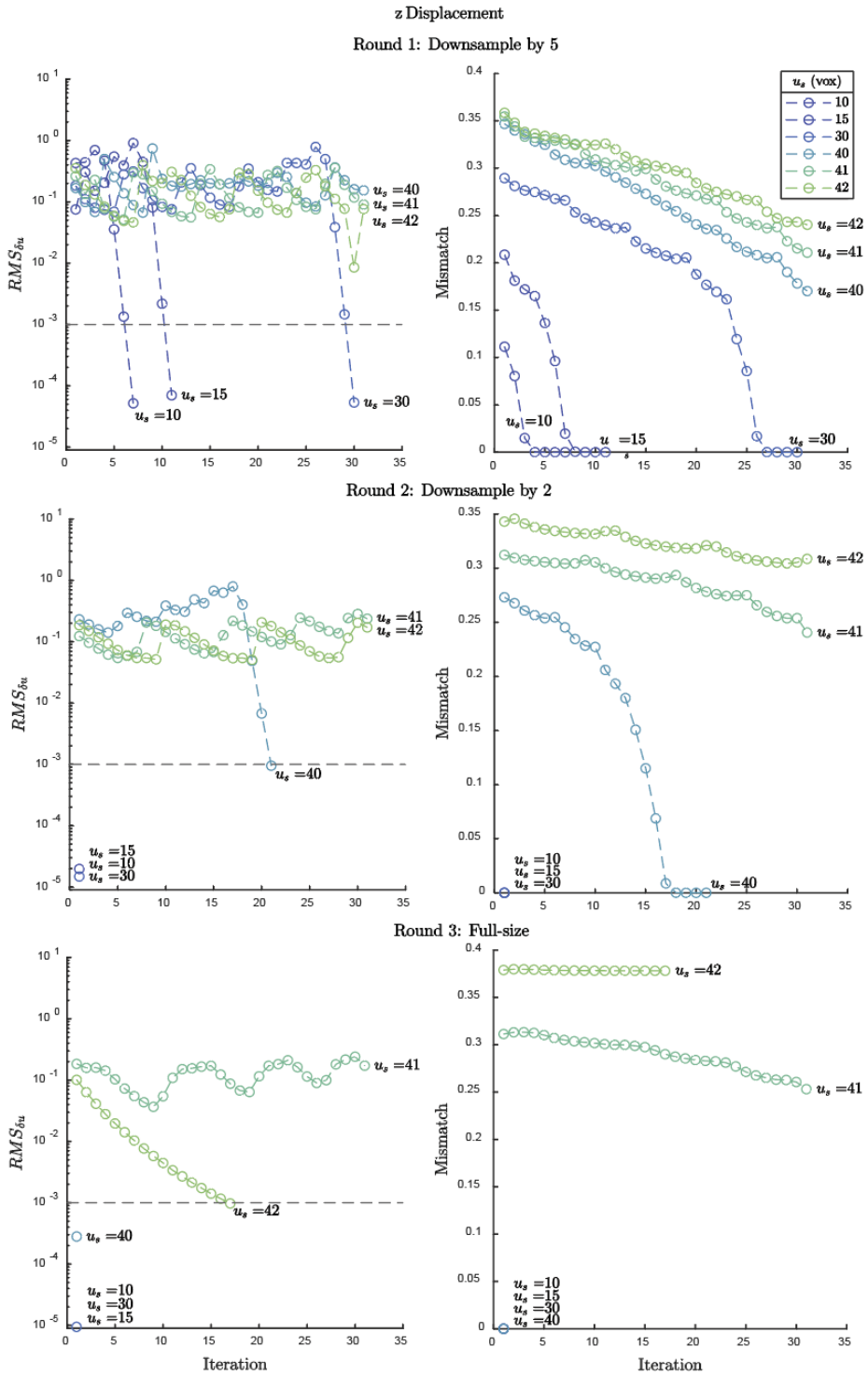


Figure 5.13: $RMS_{\delta u}$ and mismatch plots at each round for the z-displacement case. The u_s values shown are relative to the full-size image and scaled down for the earlier rounds. Isotropic voxel size for each scale was 0.185mm for downsample by 5, 0.074 for downsample by 2, and 0.037 for full-size.

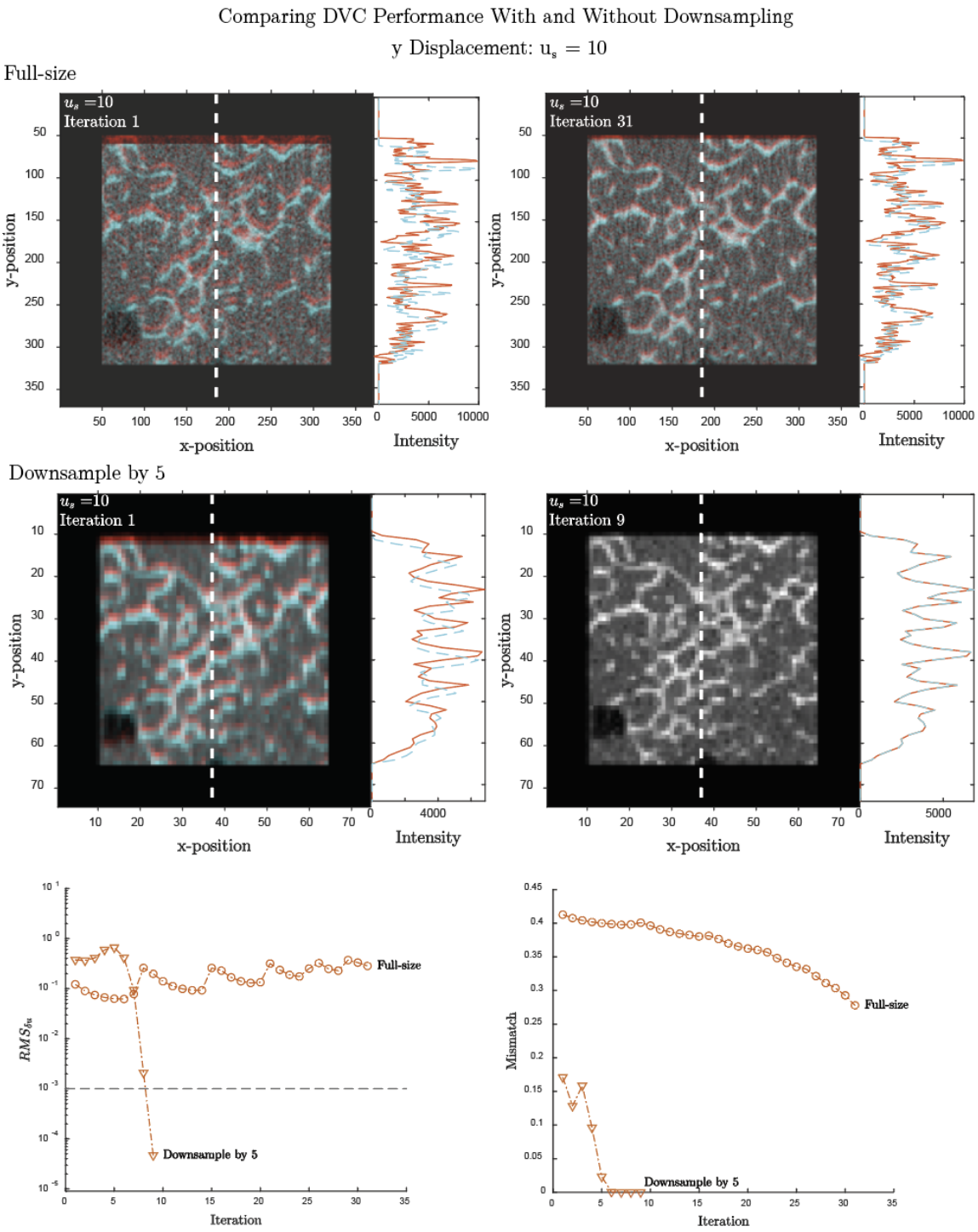


Figure 5.14: Convergence behavior is vastly improved when downsampling. Doing so allows us to converge in fewer iterations (see $RMS_{\delta u}$ plot in bottom left) at a mismatch value of zero (see Mismatch plot in bottom right). Isotropic voxel size for each scale was 0.185mm for downsample by 5 and 0.037 for full-size.

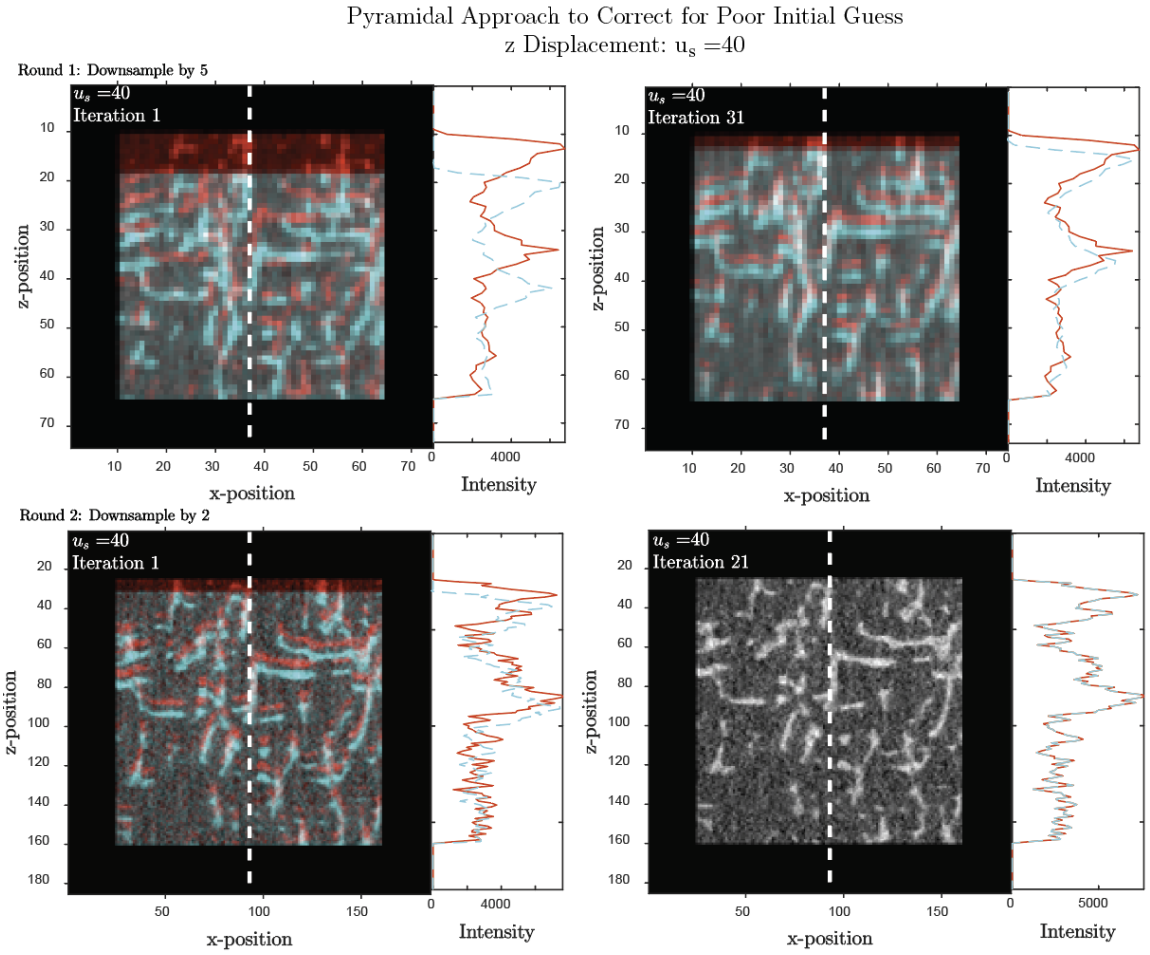


Figure 5.15: Downsampling enables us to converge at the correct displacement field at a much higher u_s than what we saw in section 5.2.1. When $u_s = 40$ in the z-displacement case, the first round helped set up the second for success, and did so with low computational cost. Isotropic voxel size for each scale was 0.185mm for downsample by 5 and 0.074 for downsample by 2.

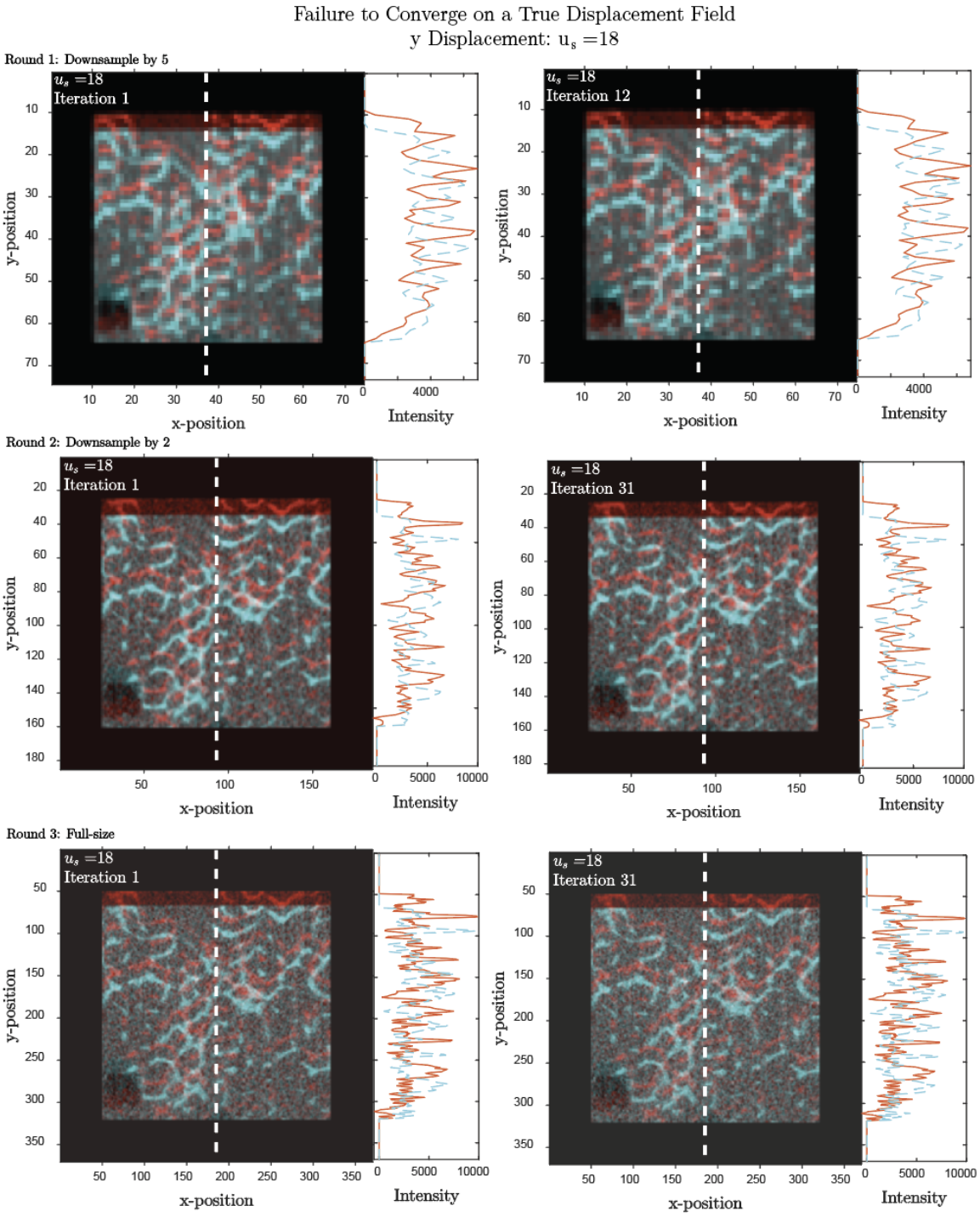


Figure 5.16: Even with our downsampling approach, there may be initial guesses that are still insufficient for DVC to execute successfully. When $u_s = 18$ in the y-displacement case, pyramidal DVC fails to calculate a displacement field that can bring the image pair closer together. Isotropic voxel size for each scale was 0.185mm for downsample by 5, 0.074 for downsample by 2, and 0.037 for full-size.

Now, let us tie our findings back to our discussion on inter-peak distance and initial guesses (at the end of section 5.2.1). In downsampling, we conglomerate intensity peaks (see section 1.2.7). As we do so, we eliminate many of the intermittent peaks that would serve as pitfalls for converging at a local minimum and widen our peaks relative to our initial guess. In this way, our "poor" initial guess is sufficient to promote quick convergence on the true displacement field. To illustrate this further, let us look at Figure 5.16 where we downsample by a factor of five in the first round (top row). Here, we have scaled our initial guess error from $u_s = 18$ (full-size) voxels to $u_s = 3.6$ (downsampled) voxels, decreased our average inter-peak distance to $\approx 5 - 10$ (downsampled) voxels, and maintained our peak width at $\approx 5 - 6$ (downsampled) voxels. In doing so, our initial guess error is within our ideal distance as defined by the peak width and is roughly less than half of the inter-peak distance.

For a more general application, how do we determine the ideal downsampling factor to start with in a pyramidal approach? The first step would be to obtain a measure of average peak width and inter-peak distance (*e.g.*, for bone these are Tb.Th and Tb.Sp, respectively). We have shown that an initial guess must at least be within half of the inter-peak distance to converge on the global minimum or, even better, less than the peak width to speed up convergence. Assuming we derive our initial guess with a rigid registration, the next consideration is how much non-rigid displacement one would expect between the images of interest. One could possibly estimate it based on the amount of displacement applied during experimentation. This would serve as a great estimate for that maximum error in the initial guess. From here, we would want a downsampling factor that will widen the

peaks and increase the inter-peak distance such that the (downsampled) initial guess error is smaller than the former and less than half of the latter. Thus avoiding local minima and promoting quick convergence. However, we must take into account uncertainties from image noise and a less than perfect rigid registration, and so we should consider a more conservative downsampling factor.

As a simple example of the above process, let us say we have a sample with typical peak width and inter-peak distance of six and 15 voxels, respectively. If our maximum, expected non-rigid, displacement is approximately eight voxels, we can assume that our initial guess derived from a rigid registration will be off by about that, eight voxels. This estimate of our error in the initial guess is larger than the peak width and half of the inter-peak distance, so we would want to downsample. After doing so, say by a factor of two, we would compare the error, now four (downsampled) voxels, to the inter-peak distance and peak width in the downsampled image and downsample further if needed.

To summarize, we have demonstrated that with our pyramidal approach, we provide DVC with a higher chance of success even when provided with a sub-optimal initial guess. Also, if we are provided a good initial guess, we still benefit from improved computational efficiency. While, there is a limit to how poor the initial guess can be, the pyramidal approach affords much greater freedom to stay below the convergence failure threshold.

6. DVC VERIFICATION AND VALIDATION

Here we will discuss verification and validation of our codes and methods. By "verification" we mean to verify that our DVC code works as intended. Our DVC code is intended to warp one image into another. To verify our code, therefore, we confirm in several test cases that the pre- and post-deformation images do indeed match after DVC. The term "validation" on the other hand, is application-specific. Here we intend to use the displacement field inferred by DVC as a measurement of a true, or physical, displacement field that occurred between two image acquisitions. To validate our code for this application, therefore, we need pre- and post-deformation images corresponding to a known displacement field. We then apply DVC to the given images and compare the resulting displacement field to the *a priori* known displacement field.

6.1. Motivation

Methods to validate algorithm performance (accuracy and precision), can fall into two categories: experimental and artificial. Experimental methods allow us to consider the contribution of noise to DVC accuracy. At present, the primary method of experimental validation is a zero-strain evaluation: researchers image a specimen twice without any deformation and calculate the mean and standard deviation of the DVC-measured displacement. Some attempts beyond zero-strain evaluation are worth noting: Franck et al. used uniaxial compression on an agarose gel sample to verify DVC capability to handle large deformations [18]; and Germaneau et al. prescribed rigid motion and made an attempt to impose homogeneous strain on a transparent material [9]. A limitation of experimental methods of validation is that they do not assess known, non-affine

deformations.

Artificial validation methods consist of artificially warping a single image according to a known displacement field. For example, researchers have applied rigid displacements: Smith et al. applied an artificial rotation [1] and Roux et al. applied half-voxel displacements [13]. Beside rigid body motion, Jandejsek et al. simulated compression, shear and torque to their images [19]. Artificial noise, such as Gaussian white noise, is often added to the deformed image in order to model the effect of imaging system noise. A limitation of purely artificial validation methods is that the contribution of noise is necessarily artificial and may be unrepresentative of that encountered in practice. We have yet to see a validation method that can account for both image noise and deformations that are nontrivial and known with a high degree of accuracy.

In this chapter we show how an independently developed image warping code can be used with repeated scans to validate our DVC approach with nontrivial, non-rigid deformation fields and realistic nonreproducibility in the image (*i.e.*, noise, air bubbles, and rigid translation from scanner). This procedure includes validation of the L-curve method to select the regularization parameter. The image warping code also provides an independent verification of image matching from DVC.

6.2. Overview of Image Warping Code

The image warping code we use for both our verification and validation methods was developed independently and will be described in detail in a forthcoming manuscript. For now, we provide a brief overview.

Overall, this code takes a given nodal displacement field and interpolates this displacement to each voxel of the image we are trying to warp. We then resample this image using this voxel displacement field to create a synthetic warped image. Our DVC code uses isoparametric finite element interpolation on hexahedral elements. Hence all displacement fields are defined on the parent domain. In order to evaluate the displacement of an individual voxel, the coordinates of that voxel location must be determined in the parent domain. For a hexahedron, the isoparametric mapping is non-linear. Therefore inverting this mapping to solve for the voxel location would require the solution of a nonlinear system of equations at every voxel location. Instead, therefore, the image warping code remeshes each hexahedron into six tetrahedra. On each tetrahedron, the isoparametric mapping is linear and thus can be efficiently inverted.

6.3. Verification Methods

When verifying our DVC algorithm, the question we ask ourselves is: are we successfully matching the given images? We sought to answer that question by using the image warping code. First, we start with two images, one pre-deformation and one post-deformation. Then we run DVC to obtain an inferred displacement field. Then we use the independent warping code to warp the post-deformation image according to the DVC deformation. Finally, we can then verify the image matching both qualitatively and quantitatively by comparing the pre-deformation and warped post-deformation images. Visualizing the difference between the warped pre-deformation image and the post-deformation image gives a qualitative sense of where the matching is successful. Second, we quantitatively measure successful matching using the "mismatch" calculation first introduced in chapter 1 (see Equation

(1.20)), i.e. we compute the overall voxel-by-voxel difference between the two images.

In a given application, we often find that the DVC code has not successfully matched a pair of images. The success depends strongly on the accuracy of the initial guess.

We explored this question in the previous section (5.2). In our exploration, we produced artificially shifted images and both created the warped image and calculated the mismatch value from the DVC results. We showed that when using a sufficiently accurate initial guess, our algorithm will correctly match a pair of rigidly shifted images. A synthetic image produced by our image warping code adds a new verification tool.

6.4. Approach for Validation of Displacement Field

The obvious follow-up question is: does our algorithm accurately calculate the displacement field for a pair of realistically deformed images? In answering this question, *i.e.* in validating our method, the primary hurdle has been creating realistically deformed images for which the displacement field is known. Our image warping code enables us to create such images.

6.4.1. Creation of non-rigidly displaced images for validation

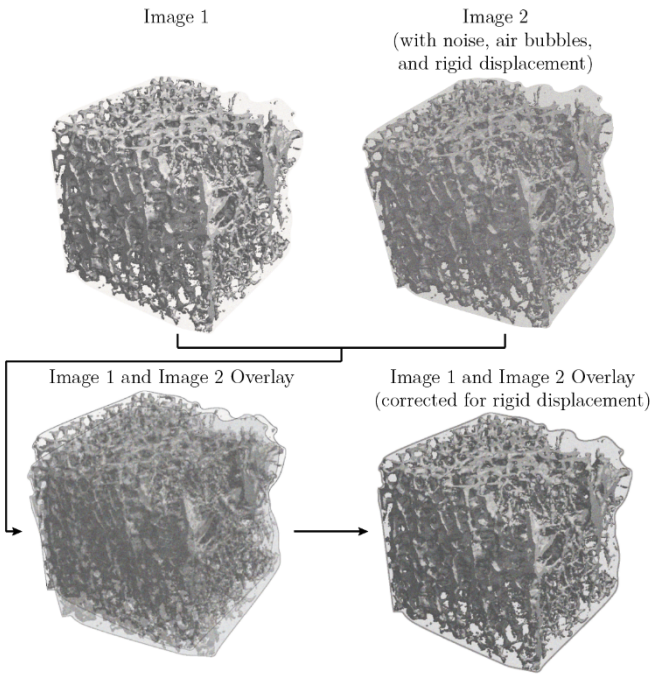


Figure 6.1: Illustration of the first step in creating our validation images. Image 1 and image 2 differ because of noise, air bubbles and rigid displacements (exaggerated for this figure). We rigidly register the two images to correct for rigid displacements.

To make a pair of images that closely mimic the type of non-rigidly displaced images we expect in an experimental, compression-tested vertebra, we must introduce image impurities such as those that are present when scanning a specimen multiple times. This is accomplished with a repeat scan: we scan a vertebra twice, where in between scans we remove and replace the sample with no load applied. This pair of images differ because of noise, formation and/or movement of air bubbles, and occurrence of microscale, rigid. We then run DVC on this image pair with high strain regularization ($\alpha = 10^{20}$ in equation (5.1)). This effectively performs a rigid registration of the two images. We then rigidly warp one of these according to the rigid displacement field just computed. The result is a pair of images that are nominally aligned, and whose intensities differ by a combination of imaging system noise, image interpolation error, and repeatability artifacts (defined in section 2.2). This is a necessary first step (see Figure 6.1) to establish any known displacement field for validation.

The next step is to produce a diverse set of non-rigid displacement fields. We created four displacement fields: uniform compression; nonuniform, that is, nonlinear compression; homogeneous shear; and torsion (see Figure 6.2).

For uniform compression, we defined our displacement field only along the z-axis as:

$$\mathbf{u} = \beta a(h - z)\,\hat{\mathbf{k}} \tag{6.1}$$

where z is the nodal z-coordinate (here, the origin is at a top corner of the cube), β is a scaling factor ($\beta = 1, 2, 4, 8,$ and 16), h is the height of the specimen, and a is a constant that makes $\mathbf{u} = 0.037\,\hat{\mathbf{k}}$ at the top surface when $\beta = 1$. When $\beta = 1$ and $\beta = 16$, the maximum magnitudes of $\mathbf{u}$ from equation (6.1) were 0.037 mm, and 0.592 mm, respectively. Note that according to this same equation, the displacements at the bottom surface of our specimen will equal zero.

For the nonlinear displacement field, we used a finer mesh and defined our displacement field only along the z-axis as:

$$\mathbf{u} = \beta\gamma\sqrt{xy}/z\;\hat{\boldsymbol{k}} \tag{6.2}$$

where $\gamma = 3/8$, and x, y, and z are the x, y, and z nodal coordinates (here again, the origin is at a top corner of the cube). When $\beta = 1$ and $\beta = 16$, the maximum magnitudes of $\mathbf{u}$ from equation (6.2) were 0.074 mm, and 1.184 mm, respectively.

For homogeneous shear in the y-z plane, we defined a displacement field along the y-axis that decreased as we go down our cubic sample:

$$\mathbf{u} = \beta a(h - z)\;\hat{\mathbf{j}} \tag{6.3}$$

where a is a constant that makes $\mathbf{u} = 0.037\,\hat{\mathbf{j}}$ at the top surface when $\beta = 1$. Again, when

$\beta = 1$ and $\beta = 16$, the maximum magnitudes of $\mathbf{u}$ from equation (6.3) were 0.037 mm, and 0.592 mm, respectively, and displacements at the bottom surface equaled zero.

For torsion about an axis located at the center of the specimen and in the z-direction, rotation by an angle $\varphi(z)$ was applied:

$$\mathbf{u} = \beta[(x_c \cos\varphi - y_c \sin\varphi - x_c)\hat{\imath} + (y_c \cos\varphi + x_c \sin\varphi - y_c)\hat{\jmath}] \quad (6.4)$$

where x_c and y_c are the nodal positions relative to the center of the specimen, $\varphi(z) = \theta(1 - z/h)$, θ was set at 0.425°, z is the distance from the top surface of the specimen, and h is the specimen height. When $\beta = 1$ and $\beta = 16$, the maximum value of the magnitude of $\mathbf{u}$ was 0.052 mm and 0.838 mm, respectively.

We warp an image from the aligned pair with one of these non-rigid, displacement fields to produce our synthetic, post-deformation image to be used as input in DVC. We measure displacement error by comparing the DVC output displacement field, $\hat{u}$, to the one we input in the image warping code, u, using a normalized root-mean-

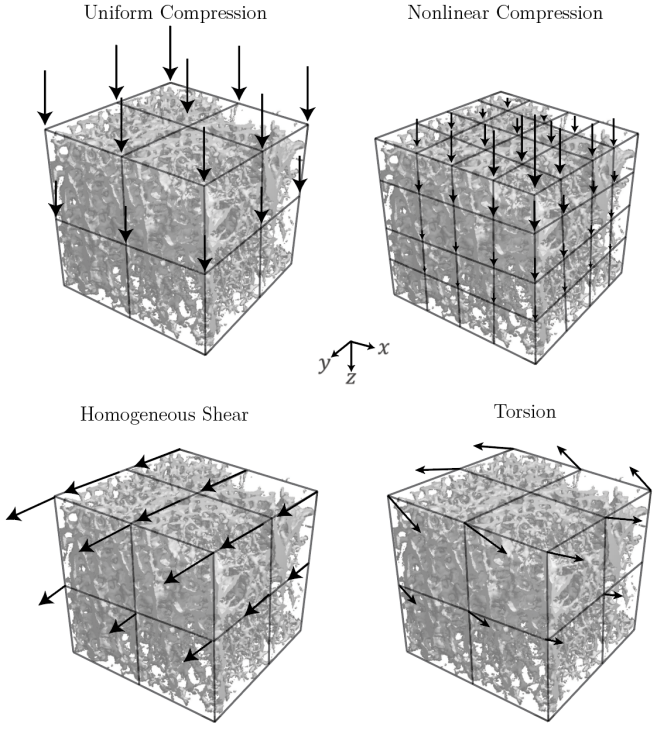


Figure 6.2: : Illustrations of the four non-rigid displacement fields we use to create our validation images. The arrows (scaled up for easy viewing) point in the direction of displacement. Note that for nonlinear compression, a finer mesh was used.

square error (NRMSE) of displacement, u_{RMSE}:

$$\mathrm{u_{NRMSE}} = \frac{\sqrt{\frac{1}{\mathrm{n_{nodes}}}\sum_{\mathrm{j=1}}^{3}\sum_{\mathrm{i=1}}^{\mathrm{n_{nodes}}}\left(\hat{\mathrm{u}}_{\mathrm{i,j}} - \mathrm{u}_{\mathrm{i,j}}\right)^2}}{\sqrt{\frac{1}{\mathrm{n_{nodes}}}\sum_{\mathrm{j=1}}^{3}\sum_{\mathrm{i=1}}^{\mathrm{n_{nodes}}}\left(\mathrm{u}_{\mathrm{i,j}}\right)^2}} \tag{6.5}$$

In equation (6.5), we sum over squared error of the x-, y- and z-displacements for each node denoted as subscript/index i for the total number of nodes, n_{nodes}.

In a similar fashion, we calculate NRMSE for infinitesimal strain, $\boldsymbol{\epsilon}$:

$$\boldsymbol{\epsilon}_{\mathrm{NRMSE}} = \frac{\sqrt{\frac{1}{\mathrm{n_{elem}}}\sum_{\mathrm{j=1}}^{6}\sum_{\mathrm{i=1}}^{\mathrm{n_{elem}}}\left(\hat{\epsilon}_{\mathrm{i,j}} - \epsilon_{\mathrm{i,j}}\right)^2}}{\sqrt{\frac{1}{\mathrm{n_{elem}}}\sum_{\mathrm{j=1}}^{6}\sum_{\mathrm{i=1}}^{\mathrm{n_{elem}}}\left(\epsilon_{\mathrm{i,j}}\right)^2}} \tag{6.6}$$

With ϵ_{NRMSE} however, we sum over the squared error of each strain component for the total number of elements, n_{elem}.

6.5. Approach for Validation of Regularization Parameter Selection Method

6.5.1. Selection of the regularization parameter

With our regularization approach using the regularization parameter, α in equation (5.1), we must make sure to use an optimal value that will mitigate the effects of noise while also providing an accurate measure of displacement. Ideally, we would find the α that produces the lowest RMSE or NRMSE. However, in a realistic experiment, the true displacement field is unknown and thus the RMSE and NRMSE cannot be computed. Instead, a typical, surrogate approach used to estimate an appropriate value for α is an L-curve [20]. With this approach, we plot our primary cost (i.e. image match cost) versus regularization cost

to form an "L"-shaped curve. The turning point of that curve is considered to be an appropriate choice for α. We want to validate this approach for our cost function.

To validate, we first need a "gold standard"; knowledge of the truly optimal α. We can do so by evaluating u_{RMSE} as a function of α, where the optimized α corresponds to the value that produces the lowest error.

With the optimal regularization parameter known, we can confirm whether the α at the turning point of the L-curve is at or near the optimal value. To build our L-curve, we compare image match and regularization costs defined as, respectively:

$$Image\ Match\ Cost = \frac{1}{2}\int_{\Omega}\left[I_1(\boldsymbol{x}) - I_2\big(\boldsymbol{x}+\boldsymbol{u}(\boldsymbol{x})\big)\right]^2 d\Omega \tag{6.7}$$

$$Regularization\ Cost = \frac{1}{2}\int_{\Omega}\boldsymbol{\epsilon}:\boldsymbol{\epsilon}\, d\Omega \tag{6.8}$$

6.6. Application of Verification and Validation Approaches

To put our verification and validation approaches to work, we created test images with the displacement fields described in chapter 6 with varying average displacements ($\beta = 1, 2, 4, 8,$ and 16 voxels) and regularization parameter values ($\alpha = 10^1, 10^2, 10^3, \dots, 10^{10}$). We first start by building our displacement and strain error plots to compare with the corresponding L-curves as a way to validate the surrogate use of the L-curve in determining the optimal α. Then, at that optimal value of α, we view our quantitative (match improvement, as defined by a decrease in mismatch value) and qualitative (image warping) measures to confirm successful image matching, i.e. verify DVC. Finally, the NRMSE value validates the use of DVC for measuring non-rigid displacement fields.

The error plots in Figure 6.3 show that the truly optimal value of α is either 10^7 or 10^8 for larger displacement magnitudes ($u = 8$ or 16) or 10^8 or 10^9 for lower displacement magnitudes ($u = 1, 2,$ or 4). Meanwhile the L-curve, regardless of displacement magnitude or type of non-rigid displacement, suggests that 10^8 is a good choice. Thus, the L-curve method provided, at best, the optimal α and, at worst, an α that provides an error that is on the same order of magnitude as the truly optimal value.

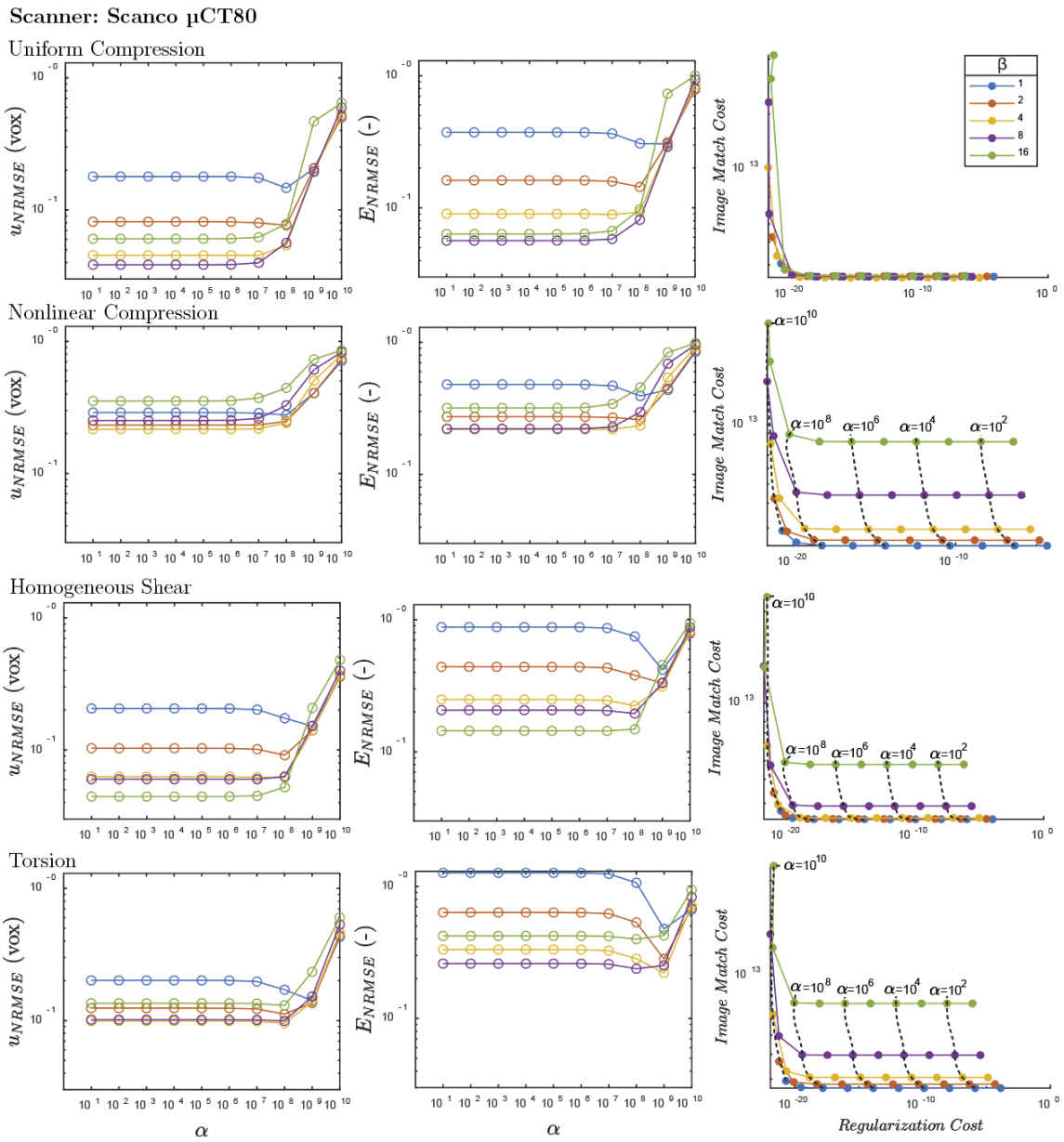


Figure 6.3: From the images scanned with the Scanco μCT80 scanner, for each loading scenario: uniform compression (first row), nonlinear compression (second row), homogeneous shear (third row), and torsion (fourth row), we present the u_{NRMSE} (left), E_{NRMSE} (center), and L-curve (right) plots.

Scanner: Scanco μCT80

Match Improvement

Pre- → Post-DVC	$\boldsymbol{\beta = 1}$	$\boldsymbol{\beta = 2}$	$\boldsymbol{\beta = 4}$	$\boldsymbol{\beta = 8}$	$\boldsymbol{\beta = 16}$
Uniform Compression	0.031→0.020	0.058→0.020	0.109→0.020	0.185→0.021	0.277→0.022
Nonlinear Compression	0.023→0.019	0.037→0.020	0.065→0.023	0.112→0.032	0.179→0.051
Homogeneous Shear	0.031→0.020	0.062→0.020	0.129→0.020	0.226→0.022	0.333→0.030
Torsion	0.025→0.022	0.046→0.024	0.094→0.025	0.172→0.032	0.272→0.053

Table 6.2: From Scanco images, match improvement values for each displacement case at each magnitude when $\boldsymbol{\alpha = 10^8}$.

At $\alpha = 10^8$, we see that DVC improves image match (see Table 6.2), which is seen as a decrease in mismatch value. In all instances, there is match improvement. Table 6.1 provides the u_{NRMSE} values for each displacement type and magnitude when $\alpha = 10^8$.

Scanner: Scanco μCT80

$\boldsymbol{u_{NRMSE}\ (\alpha = 10^8)}$	$\beta = 1$	$\beta = 2$	$\beta = 4$	$\beta = 8$	$\beta = 16$
Uniform Compression	0.1469	0.0765	0.0540	0.0564	0.0786
Nonlinear Compression	0.2772	0.2485	0.2417	0.3277	0.4441
Homogeneous Shear	0.1739	0.0914	0.0619	0.0630	0.0524
Torsion	0.1711	0.1119	0.0951	0.0993	0.1303

Table 6.1: From Scanco images, normalized error measures at $\boldsymbol{\alpha = 10^8}$.

In this table, we see that across a displacement field type, such as uniform compression, there exists an ideal displacement range where we have the lowest error. For uniform compression and torsion, the lowest error occurs between $\beta = 4$ (displacements of 0-0.148 mm) and $\beta = 8$ (displacements of 0-0.296 mm). For homogeneous shear and nonlinear compression, the lowest error was found at the highest β. Across the board, nonlinear compression presented higher error. What this shows is that there exists an ideal displacement range where we maximize the signal-noise-ratio, the signal here being the

true displacement field, u. Therefore, when designing an experiment, we should strive to apply a displacement field with a magnitude that maximizes this ratio. Qualitative results are also included for each loading scenario (see Figure 6.4) and they reveal good matching for all except nonlinear compression. There is a slight misalignment at the top end of the image. The higher error seen both quantitative and qualitative measures in nonlinear compression are expected as we use a linear interpolation for a nonlinear displacement.

All of these above results pertain to images provided by the Scanco μCT80 scanner. As we moved to a newer scanner, Zeiss Xradia Versa (Xradia, Pleasanton, CA), we wanted to explore the same results. Some noteworthy characteristics of the images captured by the Xradia is that the intensity values are stored as unsigned 16-bit integers, where Scanco's values were signed 16-bit. Therefore, Xradia's intensity values were of a much higher magnitude. Validation of DVC with Xradia images showed some promising results. First, on the subject of finding the ideal value of α (see Figure 6.5), the L-curve pointed to 10^9, which is higher than Scanco's α. This is because the image match cost is an order of magnitude higher than Scanco's due to the difference in integer type. An α of 10^9, just like in Scanco, provided relatively low displacement error. A review of match improvement (see Table 6.3), reveals consistent success in matching the non-rigidly displaced images with DVC.

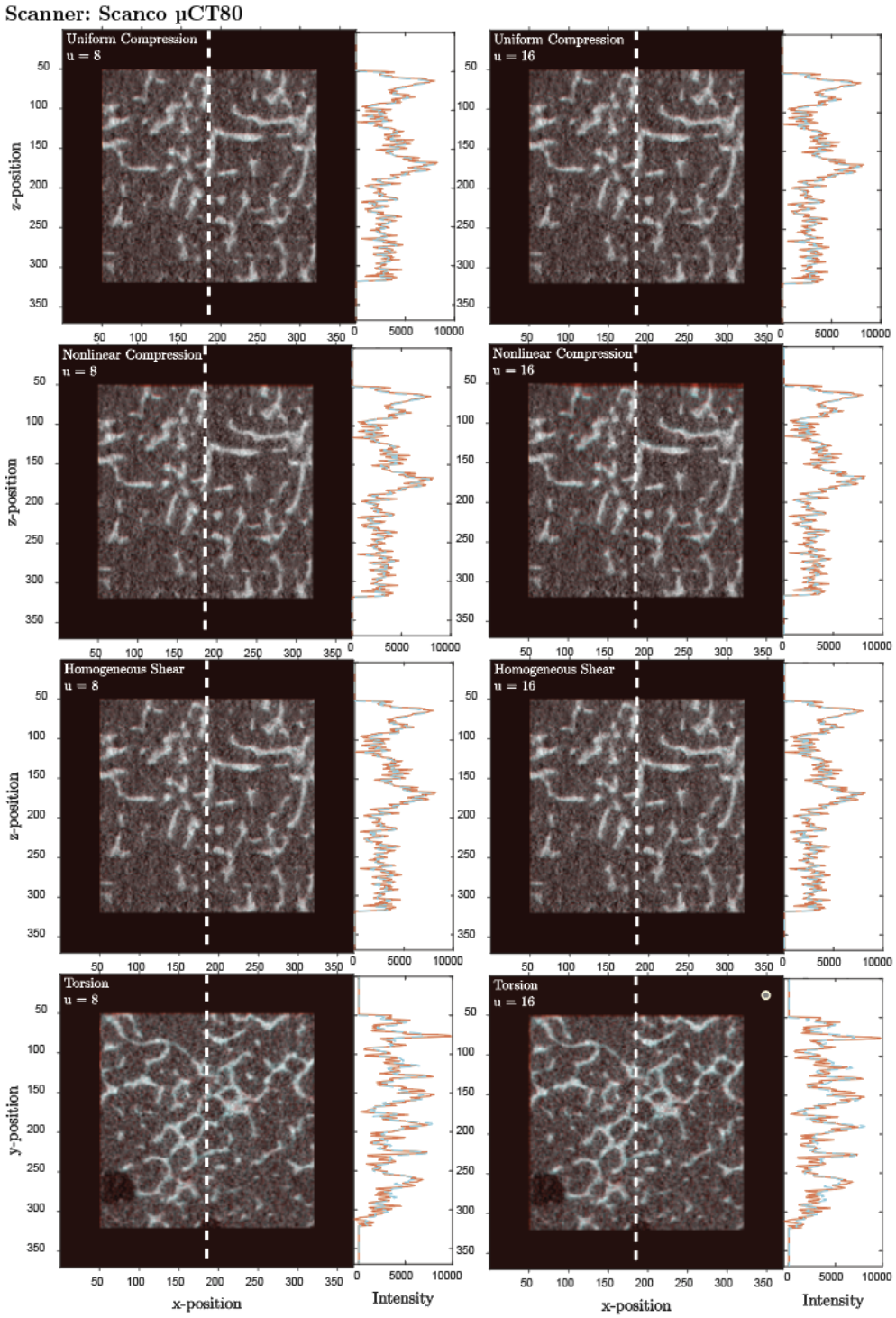


Figure 6.4: For Scanco images, qualitative review of image matching performance for our various, non-rigid displacement types and the two highest magnitudes (β=8 and 16) while using $\alpha = 10^8$. For most of these cases, there is good qualitative matching. Poor performance is seen in the top right region of the image for the high-displacement (β=16), nonlinear compression case. This is corroborated by a high u_{NRMSE} value seen in Table 6.1.

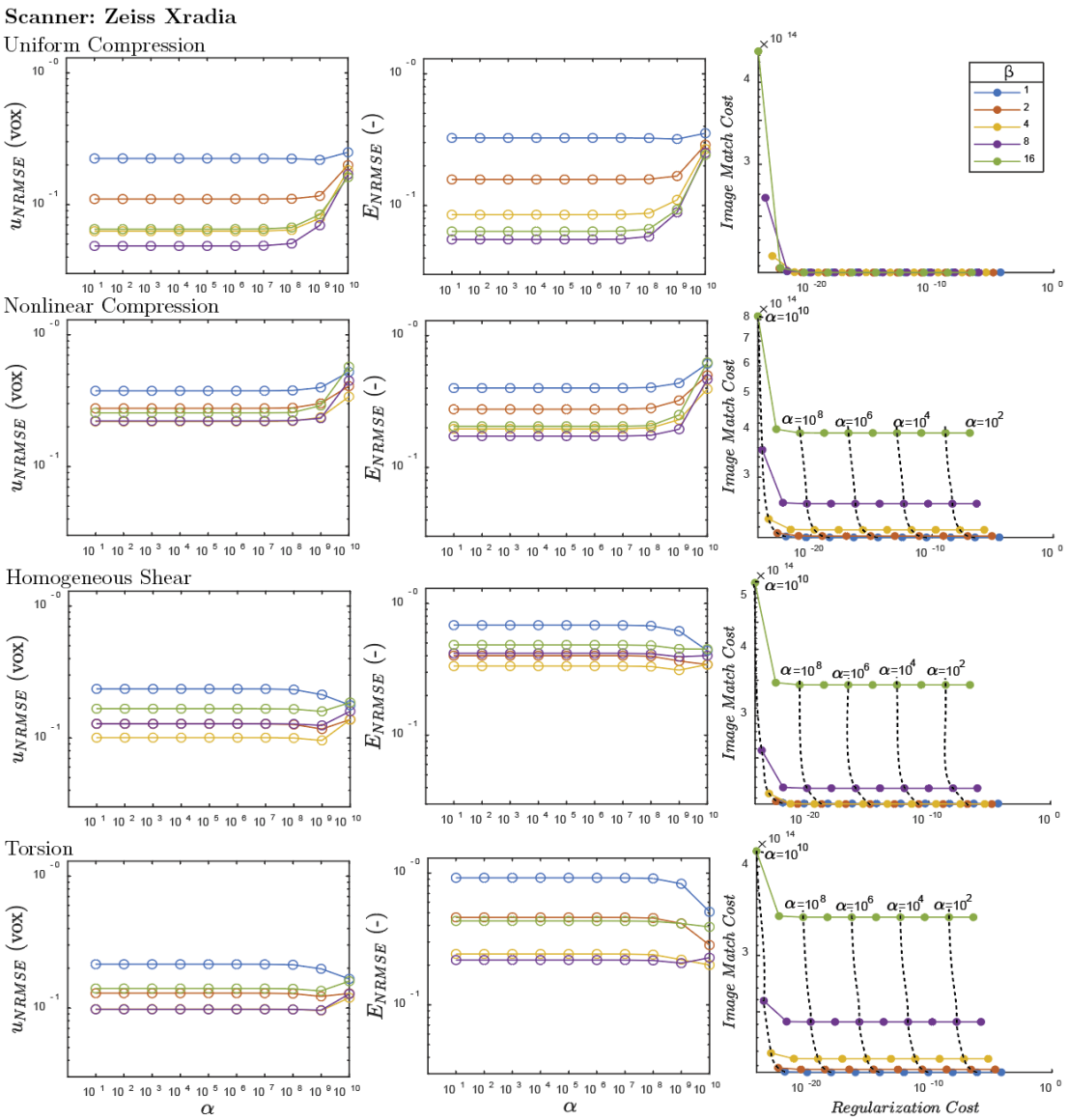


Figure 6.5: From the images scanned with the Zeiss Xradia scanner, for each loading scenario: uniform compression (first row), nonlinear compression (second row), homogeneous shear (third row), and torsion (fourth row), we present the u_{NRMSE} (left), E_{NRMSE} (center), and L-

Scanner: Zeiss Xradia

Match Improvement

Pre- → Post-DVC	$\boldsymbol{\beta = 1}$	$\boldsymbol{\beta = 2}$	$\boldsymbol{\beta = 4}$	$\boldsymbol{\beta = 8}$	$\boldsymbol{\beta = 16}$
Uniform Compression	0.011→0.007	0.018→0.007	0.039→0.007	0.079→0.007	0.136→0.008
Nonlinear Compression	0.010→0.007	0.014→0.008	0.024→0.009	0.046→0.011	0.083→0.018
Homogeneous Shear	0.012→0.007	0.021→0.007	0.048→0.007	0.102→0.008	0.168→0.013
Torsion	0.010→0.007	0.018→0.008	0.037→0.008	0.079→0.010	0.141→0.016

Table 6.3: From Xradia images, match improvement values for each displacement case at each magnitude when $\boldsymbol{\alpha = 10^9}$.

Second, similar to what we had observed with Scanco images, the displacement scaling factors of $\beta = 4$ and $\beta = 8$ provided some of the lowest errors (see Table 6.4).

Scanner: Zeiss Xradia

$\boldsymbol{u_{NRMSE}}$ $\boldsymbol{(\alpha = 10^9)}$	$\boldsymbol{\beta = 1}$	$\boldsymbol{\beta = 2}$	$\boldsymbol{\beta = 4}$	$\boldsymbol{\beta = 8}$	$\boldsymbol{\beta = 16}$
Uniform Compression	0.2186	0.1164	0.0789	0.0697	0.0841
Nonlinear Compression	0.3979	0.3012	0.2366	0.2324	0.2905
Homogeneous Shear	0.2131	0.1168	0.0957	0.1245	0.1590
Torsion	0.1981	0.1222	0.0954	0.0962	0.1346

Table 6.4: From Xradia images, normalized error measures at $\boldsymbol{\alpha = 10^9}$.

Thirdly, a qualitative review of the matched images (see Figure 6.6) supported our measures of match improvement. It is worth noting, however, that, at least qualitatively, the Xradia images appear to provide smoother images (less grain, or less noise) and a much clearer contrast between the marrow and trabecular bone. This likely caused the displacement error to be smaller than Scanco's when applying larger displacement fields ($\beta = 8$ or 16).

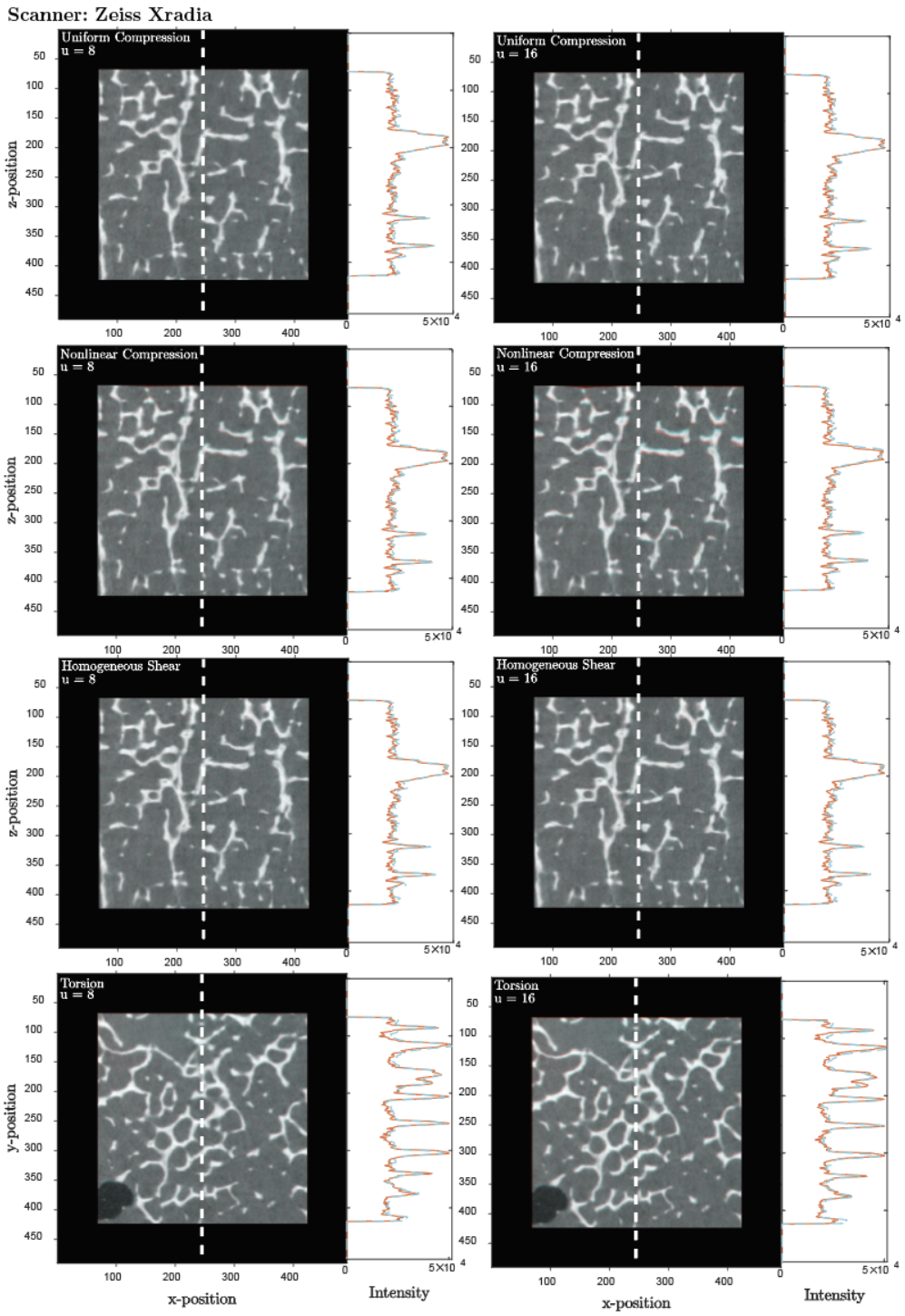


Figure 6.6: For Xradia images, qualitative review of image matching performance for our various, non-rigid displacement types and the two highest magnitudes (β=8 and 16) while using $\alpha = 10^8$. For most of these cases, there is good qualitative matching. Poor performance is seen in the top region of the image for the high-displacement (β=16), nonlinear compression case.

7. DISCUSSION AND CONCLUSION

7.1. Results Summary

In our presentation of DVC in 1D, we provide an accessible explanation of our DVC algorithm and its characteristics. We showed the use of regularization as a way to correct for the ill-posed problem of image matching. What makes this problem ill-posed is the fact that regions where intensity is uniform ($\partial I/\partial x$ =0), any displacement field works. Furthermore, with image noise being present, a non-regularized algorithm would incorrectly match noise as opposed to the true intensity signal. Therefore, in our algorithm, we regularize through both a dedicated term in our cost function and spatial averaging. We also explored the importance of providing an initial guess that places corresponding intensity peaks between images close to one another. We demonstrated that failure to do so creates a risk of converging on the wrong displacement field. To address this limitation, we downsampled our 1D images to decrease intensity peak distance. Not only does downsampling improve the effectiveness of the initial guess, but it also improves algorithm efficiency.

Before transferring this knowledge to 3D, we provide an overview of image acquisition settings and necessary data preparation for using DVC on the human vertebrae. This provides the reader a skeleton of an approach to follow in a separate study.

In our presentation of DVC in 3D, we help decipher the complexity of going from 1D to 3D and demonstrate that the characteristics explored in 1D apply in 3D as well. In 3D we are able to tie convergence behavior to physical traits of the material under assessment. For

example, for bone, Tb.Th and Tb.Sp are measures that help inform how “close” an initial guess needs to be and how downsampling can lower the bar on how close is close enough.

In our algorithm, we obtain an initial guess by rigidly registering the images of interest. This rigid registration can either be performed using the moments of mass approach presented or, effectively, by running DVC with high penalization/regularization of non-rigid displacement fields. The latter, however, itself requires an appropriate initial guess that the moments method does not.

Finally, we present new methods for verification and validation that we developed for our algorithm. A necessary first step was to develop an image warping code, which was provided independently. Using this code, we verified that our DVC algorithm matches one image to another through development of quantitative and qualitative assessments. Next, we developed a method for validating that our DVC correctly infers a realistic displacement field. A necessary sub-step in this validation was presenting an approach to validate the use of an L-curve to determine how much weight to put on the regularization term of our cost function. Then, with appropriate regularization, we were able to calculate measurement error with non-zero strain fields in repeat scans.

7.2. Comparison to Prior Work

All explanations of DVC in prior work cover DVC at a higher level, which makes the work less accessible to newcomers. Still we recommend interested readers move to more expansive review articles after studying this document. Roux et al. provides an extensive review for all applications of DVC, not just in the biological realm [21]. Roberts et al.

provides a literature review of DVC uses for measures of displacement and strain fields in bone, including a useful review of the effect of individual parameters on DVC performance [22].

On the subject of initial guesses, other methods have been used. Commercial software may be used to rigidly register two images prior to running DVC [23]. This would align the images well, but introduce interpolation error. Our method provides a way to calculate a rigid registration field without the need of commercial software or manipulating any of the input images. An alternative to using rigid registration to obtain an initial guess was demonstrated by Leclerc et al. [24] where they used a computationally efficient, though less accurate, local DVC (assumed discontinuous displacement as described in 1.2.3) to then inform a more accurate, global DVC (assumed continuous displacement).

Downsampling in conjunction with DVC to improve computational efficiency and reduce the impact of a poor initial guess is not new and is discussed by Leclerc et al. [24]. Here, they propose a pyramidal approach to improve the effectiveness of an initially poor initial guess. This was applied with local DVC by Palanca et al. [25] to leverage the computational efficiency of downsampling. In our study, we have not only proven effectiveness of a pyramidal approach in global DVC, but demonstrated its power to a more general public with our 1D and 3D illustrations.

Previous studies have estimated measurement error with repeat scans and zero-strain displacement fields [6], [26]. Those approaches, however, serve more as verification steps. Validation should mimic realistic testing conditions as closely as possible. For example,

while we reported measurement errors of 0.05-0.4 voxels using artificial deformations of a repeat scan, Jandejsek et al. reported errors as low as 0.00051 voxels on artificial deformations of the same reference scan of trabecular bone [19]. The large disparity here can most likely be attributed to the lack of noise present in their verification. Most other studies report measurement error using zero-strain tests where the mean and standard deviation of displacement and strain are used to measure accuracy and precision, respectively. For instance, Liu and Morgan reported a zero-strain, displacement precision of 0.076 voxels when using a repeat scan and an element size of 40 voxels (1.44 mm) [6]. In Hussein et al. this estimate of precision was 1.12 voxels for an element size of 135 voxels (5.00 mm) [23]. However, the precision of zero-strain displacement fields cannot be directly compared to our non-rigid displacement field approach. As an alternative, we calculated strain accuracy and precision for uniform compression using the method of Liu and Morgan and Hussein et al. (see Table 7.1) and compared these values to those in these two published studies. The image differences in both of their studies include noise, repeatability artifacts, and interpolation, while ours has all of that plus actual deformation of the intensity peaks. Even with that extra layer of complexity, we had a comparable strain accuracy and precision to both studies. More interestingly, we noted that accuracy and

Strain Accuracy and Precision

		Mean/ Accuracy (ϵ)	*Standard Deviation/ Precision (ϵ)*
Zero-strain	*Liu and Morgan*	3.45×10^{-4}	1.25×10^{-4}
	Hussein et al.	7.40×10^{-4}	6.30×10^{-4}
Uniform Compression	$\beta = 1$	5.03×10^{-4}	1.76×10^{-5}
	$\beta = 2$	9.82×10^{-4}	2.72×10^{-5}
	$\beta = 4$	2.10×10^{-3}	3.59×10^{-5}
	$\beta = 8$	4.70×10^{-3}	1.40×10^{-4}
	$\beta = 16$	1.00×10^{-2}	3.29×10^{-4}

Table 7.1: Strain accuracy and precision of the uniform compression field at different β values were calculated using the method described by Liu and Morgan [6] and Hussein et al. [23].

precision worsen with higher displacements. Thus demonstrating a need to standardize a benchmarking procedure that takes into account non-zero strains. Therefore, we recommend the use of our validation methods to establish measurement error as opposed to the verification methods previously used. This will, hopefully, promote a more consistent and understandable benchmarking tool across algorithms.

7.3. Implications

Though the advancements introduced here were developed for our specific DVC algorithm, there are several elements that are transferable to alternative algorithms. The lessons taught in our 1D exploration are applicable across dimensions and across DVC algorithms. These lessons cover different types of regularization, initial guesses, and the benefits of downsampling. We propose the use of rigid registration as a means for calculating an initial guess. We also present the use of image warping as a tool for verification and validation, which others can replicate for calculating measurement error with realistic displacement fields. We foresee this work to be foundational for the progress of DVC algorithms everywhere.

BIBLIOGRAPHY

[1] T. S. Smith, B. K. Bay, and M. M. Rashid, "Digital volume correlation including rotational degrees of freedom during minimization," *Experimental Mechanics*, vol. 42, no. 3, pp. 272–278, Sep. 2002.

[2] E. Verhulp, B. Van Rietbergen, and R. Huiskes, "A three-dimensional digital image correlation technique for strain measurements in microstructures," *Journal of Biomechanics*, vol. 37, no. 9, pp. 1313–1320, Sep. 2004.

[3] O. Jiroušek, I. Jandejsek, and D. Vavřík, "Evaluation of strain field in microstructures using micro-CT and digital volume correlation," *Journal of Instrumentation*, vol. 6, no. 1, C01039, 2011.

[4] B. K. Bay, T. S. Smith, D. P. Fyhrie, and M. Saad, "Digital volume correlation: Three-dimensional strain mapping using X-ray tomography," *Experimental Mechanics*, vol. 39, no. 3, pp. 217–226, 1999.

[5] R. Zauel, Y. N. Yeni, B. K. Bay, X. N. Dong, and D. P. Fyhrie, "Comparison of the linear finite element prediction of deformation and strain of human cancellous bone to 3D digital volume correlation measurements.," *Journal of Biomechanical Engineering*, vol. 128, no. 1, pp. 1–6, Jul. 2006.

[6] L. Liu and E. F. Morgan, "Accuracy and precision of digital volume correlation in quantifying displacements and strains in trabecular bone," *Journal of Biomechanics,* vol. 40, no. 15, pp. 3516–3520, Jan. 2007.

[7] F. Brémand, A. Germaneau, P. Doumalin, and J. C. Dupré, "Study of mechanical behavior of cancellous bone by digital volume correlation and x-ray micro-computed tomography," in *Society for Experimental Mechanics - 11th International Congress and Exhibition on Experimental and Applied Mechanics 2008*, vol. 3, pp. 1527–1533.

[8] M. R. Hardisty and C. M. Whyne, "Whole bone strain quantification by image registration: A validation study," *Journal of Biomechanical Engineering*, vol. 131, no. 6, Jun. 2009.

[9] A. Germaneau, P. Doumalin, and J. C. Dupré, "Full 3D measurement of strain field by scattered light for analysis of structures," *Experimental Mechanics*, vol. 47, no. 4, pp. 523–532, Aug. 2007.

[10] M. Gates, J. Lambros, and M. T. Heath, "Towards High Performance Digital Volume Correlation," *Experimental Mechanics*, vol. 51, no. 4, pp. 491–507, Apr. 2011.

[11] B. Pan, D. Wu, and Z. Wang, “Internal displacement and strain measurement using digital volume correlation: A least-squares framework,” *Measurement Science & Technology*, vol. 23, no. 4, 2012.

[12] H. Leclerc, J. N. Périé, S. Roux, and F. Hild, “Voxel-Scale Digital Volume Correlation,” *Experimental Mechanics*, vol. 51, no. 4, pp. 479–490, Apr. 2011.

[13] S. Roux, F. Hild, P. Viot, and D. Bernard, “Three-dimensional image correlation from X-ray computed tomography of solid foam,” *Composites. Part A, Applied Science and Manufacturing*, vol. 39, no. 8, pp. 1253–1265, 2008.

[14] H. Huang, D. Dabiri, and M. Gharib, “On errors of digital particle image velocimetry,” Measurement Science and Technology, vol. 8, no. 12, 1427, 1997.

[15] C. E. Willert and M. Gharib, “Digital particle image velocimetry,” *Experiments in Fluids*, vol. 10, no. 4, pp. 181–193, 1991.

[16] R. G. Keys, “Cubic Convolution Interpolation for Digital Image Processing,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 29, no. 6, pp. 1153–1160, 1981.

[17] H. LeVine, *Medical imaging*. Santa Barbara, Calif.: Santa Barbara, Calif. : Greenwood, 2010.

[18] C. Franck, S. Hong, S. A. Maskarinec, D. A. Tirrell, and G. Ravichandran, “Three-dimensional full-field measurements of large deformations in soft materials using confocal microscopy and digital volume correlation,” *Experimental Mechanics*, vol. 47, no. 3, pp. 427–438, Jun. 2007.

[19] I. Jandejsek, O. Jiroušek, and D. Vavřík, “Precise strain measurement in complex materials using digital volumetric correlation and time lapse micro-CT data,” in *Procedia Engineering*, 2011, vol. 10, pp. 1730–1735.

[20] C. R. Vogel, *Computational Methods for Inverse Problems*. Society for Industrial and Applied Mathematics, 2002.

[21] A. Buljac *et al.*, “Digital Volume Correlation: Review of Progress and Challenges,” *Experimental Mechanics*, vol. 58, no. 5, pp. 661–708, Jun. 2018.

[22] B. C. Roberts, E. Perilli, and K. J. Reynolds, “Application of the digital volume correlation technique for the measurement of displacement and strain fields in bone: A literature review,” *Journal of Biomechanics*, vol. 47, pp. 923–34, 2014.

[23] A. I. Hussein, P. E. Barbone, and E. F. Morgan, “Digital Volume Correlation for Study of the Mechanics of Whole Bones.,” *Procedia IUTAM*, vol. 4, pp. 116–125,

2012.

[24] H. Leclerc, S. Roux, and F. Hild, “Projection Savings in CT-based Digital Volume Correlation,” *Experimental Mechanics*, vol. 55, no. 1, pp. 275–287, May 2015.

[25] M. Palanca, G. Tozzi, L. Cristofolini, M. Viceconti, and E. Dall’Ara, “Three-dimensional local measurements of bone strain and displacement: Comparison of three digital volume correlation approaches,” *Journal of Biomechanical Engineering*, vol. 137, no. 7, Jul. 2015.

[26] E. Dall’Ara, D. Barber, and M. Viceconti, “About the inevitable compromise between spatial resolution and accuracy of strain measurement for bone tissue: A 3D zero-strain study,” *Journal of Biomechanics,* vol. 47, no. 12, pp. 2956–2963, 2014.

CURRICULUM VITAE